Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Aleksi Mustonen

Ways to improve Continuous Deployment processes

Master's Thesis
Espoo, February 24, 2020

Supervisors: Jari Vanhanen
Advisor: Nils Haglund M.Sc (Tech.)

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Aleksi Mustonen

Title:
Ways to improve Continuous Deployment processes

Date: February 24, 2020 **Pages:** vi + 80

Major: Software and Service Engineering **Code:** SCI3043

Supervisors: Jari Vanhanen

Advisor: Nils Haglund M.Sc (Tech.)

Modern software development forces companies to release high quality software faster to respond to market needs. These requirements force software companies to invest on processes which make them able to make development faster without losing quality. Continuous deployment is becoming popular practice to answer the market needs, but using it efficiently can be hard. In this thesis we investigate the benefits, problems and best practices relating to continuous deployment using a literature review and a case study. The objective of this research is to identify the benefits of continuous deployment, and to discover which problems and best practices occur in our case organization, and how often.

We investigate a case organization by using interviews and a survey for software professionals. Our main finding is that continuous deployment is the preferred way to do software development and it has numerous benefits. Our case study identified a number of problems which harm the development work. Lack of automated tests and schedule pressures were the most often happening problems. We identified numerous best practices but none of them was used often. These practices included for example techniques which improved test automation practices and organizational support methods for the development team. Future research could focus on why the problems occur often, and why best practices are not widely used.

Keywords: continuous integration, continuous delivery, continuous deployment, empirical software engineering

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Aleksi Mustonen		
Työn nimi:	Tapoja kehittää Continuous Deployment prosesseja		
Päiväys:	24. helmikuuta 2020	**Sivumäärä:**	vi + 80
Pääaine:	Software and Service Engineering	**Koodi:**	SCI3043
Valvojat:	Jari Vanhanen		
Ohjaaja:	Diplomi-insinööri Nils Haglund		

Nykyaikainen ohjelmistokehitys pakottaa yritykset julkaisemaan korkealaatuisia ohjelmistoja nopeammin vastatakseen markkinoiden tarpeisiin. Nämä vaatimukset pakottavat ohjelmistoyritykset investoimaan prosesseihin, joiden avulla ne voivat tehdä kehitystä nopeampaa menettämättä laatua. Continuous deploymentista on tulossa suosittu käytäntö vastata markkinoiden tarpeisiin, mutta sen tehokas käyttö voi olla vaikeaa. Tässä diplomityössä tutkitaan continuous deploymenttiin liittyviä hyötyjä, ongelmia ja parhaita käytäntöjä. Tutkimusmetodina on tapaustutkimus, jossa tutkimme organisaatiota haastatteluilla ja kyselyllä ohjelmistokehityksen ammattilaisille. Tämän tutkimuksen tavoitteena on tunnistaa continuous deploymentin hyödyt ja selvittää, mitä ongelmia ja parhaita käytäntöjä esiintyy tapaustutkimuksemme organisaatiossa ja kuinka usein.

Tärkein havainto on, että continuous deployment on suositeltava tapa suorittaa ohjelmistokehitystä ja sillä on lukuisia etuja. Tutkimuksemme tunnisti useita ongelmia, jotka vahingoittavat kehitystyötä. Automaattisten testien puute ja aikataulu paineet olivat yleisimmin esiintyviä ongelmia. Tunnistimme lukuisia parhaita käytäntöjä, joita ei käytetä usein. Nämä käytänteet sisältivät mm. tekniikoita, joilla testiautomaatiota voidaan tehostaa sekä organisaation tukikeinoja kehitystiimin työn tehostamiseksi. Tulevaisuuden tutkimuksessa voitaisiin keskittyä siihen, miksi ongelmia esiintyy usein, eikä parhaita käytäntöjä käytetä laajasti.

Asiasanat:	jatkuva integraatio, jatkuva toimittaminen, jatkuva julkaisu, empiirinen ohjelmistokehitys
Kieli:	Englanti

Contents

Chapter 1

Introduction

1.1 Motivation

Modern software development forces companies to release high quality software faster to respond to market needs. These requirements force software companies to invest on processes which make them able to make development faster without losing quality. Shahin et al. [30] made a systematic literature review of empirical studies on continuous integration, delivery and deployment. They reviewed 69 papers and identified approaches, tools, challenges and practices related to the subject. In the article continuous integration, delivery and deployment bring the following benefits [30]:

1. *getting more and quick feedback from the software development process and customers*

2. *Having frequent and reliable releases, which lead to improved customer satisfaction and product quality*

3. *Through continuous deployment, the connection between development and operations teams is strengthened and manual tasks can be eliminated*

These benefits are desirable by companies that want to develop software and have advantage over competitors. Continuous deployment is becoming popular practice: In the 13th annual state of agile report 35% of respondents used continuous deployment[1]. However using continuous deployment efficiently can be hard. Studying continuous deployment in real-life context can produce knowledge for software development companies on how to enable reliable automation in their software development to make processes more efficient. Investigating a company,

[1]https://www.stateofagile.com/ufh-i-521251909-13th-annual-state-of-agile-report/473508

which is working in many continuous deployment projects in various industries is a great way to gain new knowledge about continuous deployment in industry.

This thesis tries to find ways how continuous deployment practises can be improved by investigating experiences of Eficode's software professionals. This thesis provides information on which problems teams need to take into account when establishing continuous deployment, and which best practices should be used to guarantee quality in development work. Results can be used as a checklist for software development teams on how to establish benefits of continuous deployment, but also to reveal new justification for links between problems and best practices, which can be motivation for further research.

1.2 Problem statement and research questions

Adoption of continuous deployment is hard for software development teams. There are many problems which block development teams from succeeding with continuous deployment. The problems can be solved by using best practices. Our research problem is to identify the problems that hinder the benefits of continuous deployment and the best practices which solve these problems and improve the continuous deployment process. This problem is solved by answering following research questions using literature review and our case study. The research questions introduced below are answered by gathering data from literature review and from our case study. All data sources answer each research question, expect the case study interviews are not used to answer RQ3 or RQ5.

- RQ1: What are the benefits of continuous deployment?
- RQ2: Which problems happen in continuous deployment?
- RQ3: How often do the problems happen in continuous deployment projects?
- RQ4: Which best practices improve the continuous deployment process?
- RQ5: How often are the best practices used in continuous deployment projects?

1.3 Structure of the thesis

This thesis starts with background, which introduces the main aspects of terms relating to thesis in chapter 2. After that we introduce the research methods in chapter 3. After that we go through the results of our literature review: Introducing the benefits, problems and solutions relating to continuous deployment in chapter 4.

After the literature review we go through the results of the case study in chapter 5. We end the research by discussing the results in chapter 6 and making conclusions in chapter 7.

Chapter 2

Background

In this chapter we introduce the main aspects of software development which relate to continuous deployment. We begin each section by introducing the concept. Then we will explain how it relates to continuous deployment. We conclude each section with a more detailed description of the concept.

2.1 Continuous integration

Shahin et al. [30] introduced continuous integration (CI) as a widely established development practice in the software industry. In CI members of software development team integrate and merge their work frequently, at least once a day. CI includes automated software building and testing [30]. CI centralizes and automates the quality assurance of software to one location. CI build consists of predefined steps. All steps passing leads to a passing CI build. Even one failing step leads to failing CI build. Passing builds indicate the development team, that the software is in functional condition. Keeping the builds passing should be the goal of the team.

Leppänen et al. [19] investigated continuous deployment practises in Finnish industries. They mentioned that continuous deployment could be considered the next step from CI [19]. Continuous deployment pipeline is similar to Continuous Integration except after successful running the software is released. Therefore, continuous integration is an indispensable prerequisite for continuous deployment.

CI is a great tool for indicating and solving problems, where a defect occurs only in a certain environment, and developers can only state that the code worked on their local machines. The term for this kind of problem is "Works on my machine". These problems can be hard to solve, as they are noticed by the developers too late. Meyer [22] wrote an article about CI and it's tools. CI is meant to be an unbiased judge of whether a change works or not, thereby preventing the "works

on my machine problem" [22].

2.2 Continuous deployment

Savor et al. [29] researched continuous deployment practises in Facebook and OANDA. They described the key elements of continuous deployment:

1. *software updates are kept as small and isolated as reasonably feasible;*
2. *they are released for deployment immediately after development and testing completes*
3. *the decision to deploy is largely left up to the developers (without the use of separate testing teams)*
4. *deployment is fully automated.*

Continuous deployment and continuous delivery are both processes where software is deployed via delivery pipeline. Shahin et al. [30] mentioned that there is a robust debate in academic and industrial circles about defining distinguishing between continuous deployment and Continuous Delivery [30]. The differences between these two processes are, that in CD the deployment of the project happens automatically, every time the code is merged to certain branch of version control. In continuous delivery the project is deployed automatically, with a manual trigger. In this research we don't want to concentrate on the minimal difference of these two processes, and we refer to both processes as CD from now on.

CD is a popular topic of research and it is approached from many angles. CD transformation is a process, where a software team or organization transfer to use CD. These studies address the issues that make CD possible, the reasons why the CD was used, and the impact of the CD on software development over old practices.

2.3 Test automation

According to Polo et al. [26], test automation exists to point out defects in the software. It is pivotal to control and reduce the costs of testing. Automated tests need to be optimized to balance the quality needs with cost [26]. Using test automation can improve the efficiency and quality of quality assurance. Test automation plays a crucial role in CD pipeline, since the software will not be released if the tests fail. It is important to invest in test automation when implementing CD [13, 25, 30]. Automated tests are meant to ensure that the software functions correctly. Next

we go through unit tests and automated acceptance tests. These tests are most relevant regarding CD.

Rogers and Owen [27] defined purpose of unit tests as verifying that piece of code operates in accordance with developer's expectations [27]. Unit test is meant for verifying that a piece of code in software fulfills it requirements. For example methods have their own unit tests for different outcomes.

Automated acceptance tests enable quality assurance for the end to end functionality of the software. Haugset and Hanssen [14] did a case study about automated acceptance testing. They defined the desire of automated acceptance tests as to document requirements and desired outcome in a format that can be automatically and repeatedly tested [14]. Customers can also be used to define the automated acceptance tests [14] [27]. These tests are implemented with test frameworks, which simulate end users by example clicking and typing to the system. Automated acceptance tests provide helpful information about the systems functionality, but they are the hardest tests to configure since they require the whole software running in test environment. Writing automated acceptance tests require excellent skills using test framework and understanding of the whole system.

Both unit tests and automated acceptance tests can be run in parallel. It is an extremely powerful method for shortening execution times, especially for automated acceptance tests. Running tests in parallel requires more effort from the team to make the tests execute correctly. Parallel tests must work independently, and they shouldn't touch any data that another test suite is using. This might lead to failing test suite caused by execution order.

2.4 Microservice architecture

According to Villamizar et al. [33], microservice architecture divides application to pieces, which are developed and tested independently. Using cloud computing allows each microservice to be deployed and scaled independently [33]. Monolithic architecture is opposite to microservice architecture: It is an application with single codebase, which can contain multiple services. When using cloud computing, all services of the software are deployed at once. Also the scaling rules apply to all services of the software [33]. The architecture of the software affects directly for the CD process. When using microservice architecture, each microservice can have a specific CD pipeline. Applications with monolithic architectures are controlled with a single pipeline.

2.5 Version control system

Spinellis [31] described using version control to be the most important tooling improvement for the development team. The workflow of version control allows the development team to share the changes in application: Developer's private version of repository can be synchronized with the team's changes [31]. Version control system allows the members of the development team to do work simultaneously. In addition, it creates transparency over the work of teammates.

According to Spinellis [31] version control protects developers when they are making changes to the same file. Version control system will warn if the changes conflict with each other. When the changes don't include any conflicts, the version control system will unobtrusively combine the changes of the developers [31]. Combining changes and resolving conflicts is an important feature of any version control system.

Version control systems are crucial when using CD. Adams et al. [1] discussed the first step of the CD pipeline is the movement of code changes made by developers in their own development branch across their team's branch all the way up to the project's master branch. Conflicts slow down the development of software, because it requires developer effort to understand the code changes in multiple branches in order to combine them correctly. Keeping branches short-lived helps to avoid conflicts [1]. Every team that uses CD must agree on common ways to use version control system. Together with CD, version control system can also be used to rollback and deploy the software to previous version, if the newest version included defects.

2.6 Code review

According to Bacchelli et al. [2] code review is a manual inspection of source code by developer other than the author. Code review is recognized as a valuable tool for reducing software defects and improving the quality of software projects. Code reviews can contain comments or changes about code in terms of readability, commenting, consistency and dead code removal. Programmers, which participated in this research ranked these improvements as an important motivation for code reviews [2]. Version control systems allow digital code reviews, where developers are able to comment code changes and approve or decline changes.

Savor et al. [29] mentioned in their research, that code reviews are prevalent part of CD process: Because developers are fully responsible for the entire lifecycle of the software, code reviews are taken more seriously and there is far less resistance for them [29]. Digital code reviews can be added to the CD process: Code change will not be deployed before it is accepted by a peer in digital code review.

2.7 Agile development

According to Balaji et al. [3] the pros of agile development is to be able to respond to changing needs, even late in development. In agile development working software is delivered frequently, as the most important principle is customer satisfaction by giving rapid and continuous delivery of small and useful software [3]. In the modern era changes need to be made quickly in order to answer competitors and market needs. Also many projects are complex, that the requirements can't be fully known at the start of the project. Agile development answers these needs by offering a methodology where software requirements can be changed frequently.CD is a great process for agile projects: It allows the development team to deploy software changes to production environment as quickly as possible for the customer.

In contrast to agile development, waterfall method relies on solid, unchangeable requirements. Balaji et al. [3] discussed the differences between waterfall and agile methods. Waterfall method is a sequential development model which consists of predefined phases. Each time boxed phase of development proceeds in order without overlapping. In waterfall clients are not allowed to make changes in requirements. Waterfall method was recommended if the requirements are fully known [3].

2.8 Team structures in software development

Stray et al. [32] described that the agile manifesto promotes the idea of autonomous, self-managing software development team whose members work at pace that sustains their creativity and productivity. Autonomous teams have challenges, as having too many dependencies and lack of organizational support [32]. Having too many dependencies can relate that only some team members are able to do certain tasks. For example if only one member of the team is capable of deploying the software, it creates a bottleneck for the team and the team is dependent on it. Team structures are an important topic in CD, because it defines the working methods of the software development team. The working methods should support the development team to be more efficient.

Adams et al. [1] described release engineering as action, where developers code is deployed to end users in automated pipeline. Release engineers responsibilities are implementing, maintaining and operating production environments [1]. Setting up and maintaining the CD pipeline can be time consuming. Maintaining the pipeline and handling releases can be allocated to a dedicated team, which can be called "Release engineering team". Also testing and quality assurance can be dedicated to a specific team. Gmeiner et al. [13] introduced "QA team" ,

which responsibilities included the test automation and quality assurance of the software [13].

2.9 Technical debt

Kruchten et al. [17] referred to the original definition of technical debt from Cunningham [10]: *Solution, which is not optimal and making it right is postponed.* Technical debt can hinder the productivity of a software development team. Technical debt can be accumulated in all artifacts of software, such as in source code, automated test cases or documentation [17].

Agile projects are strongly linked with CD. Kruchten et al. [17] mentioned that agile projects tend to gather a lot of technical debt. The reason for this can be lack of time for designing and reflecting on longer term and lack of rigorous systematic testing. Yet technical debt can be a wise choice to gain advantage when releasing features to production, however in these cases the technical debt should not be forgotten in future [17]. Technical debt can be thought of as a shortcut for faster results. In the future it is important to return to the solution which has technical debt, and reimplement it in a more optimal way.

According to the opinions of Kruchten et al [17], reasons for technical debt include schedule pressures, lack of education and poor processes [17]. Schedule pressure might be the cause of technical debt in a situation where developers need to get the task done without focusing on quality. For example a developer might write duplicate code and not write a reusable method for it because there is not enough time. Lack of education implies that in some cases the developer might not have the skills to implement the solution in the optimal way, which leads to technical debt. For example if a developer is not familiar with a specific programming language, they might write a suboptimal implementation. If the team processes do not include peer-review practices technical debt might more easily pile up. In other words the team is lacking working methods, which help them prevent technical debt growing.

Refactoring is an activity which may help reduce technical debt. Refactoring means that a solution is re-implemented to be more clear and understandable. Mens and Tourwe [21] defined the following refactoring activities, which can be used to reduce technical debt:

1. *Identify where the software should be refactored*
2. *Determine which refactoring(s) should be applied to identified places*
3. *Guarantee that the applied refactoring preserves behaviour*

4. *Assess the effects of the refactoring on quality characteristics of the software. For example for complexity, understandability, maintainability. The refactoring process can also be evaluated: productivity, cost, effort*

5. *Maintain the consistency between the refactored program code and other software artifacts, such as documentation, design documents, requirements, specifications, tests, etc.*

Chapter 3

Research methods

In this chapter we will introduce the research methods that are used to research the subjects related to CD in previous chapter. Our study is divided into two sections: Literature review and our case study. Literature review offers broad viewpoints on CD. Our case study uses interviews and a survey to collect data for software professionals.

3.1 Case study

Runeson and Höst [28] described that case study is used when studying phenomenons in real-life context [28]. Our research method is case study because we want to study CD in real-life scenarios. Our case in this research is Eficode. The justification for this selection is that Eficode was available and it provides insightful knowledge related to CD: Eficode is a software company, which works closely with clients in different industries providing development support and knowledge to software development. Eficode has also plenty of experienced professionals in the CD field, which makes it a great selection of company for this case study. By investigating Eficode's software professionals new information about CD problems and practices can be learned.

Runeson and Höst [28] stated that triangulation increases the precision of empirical research. In data source triangulation more than one data source is used to collect the same data on different occasions [28]. We used data source triangulation in this study by using two interviewees. Methodological triangulation is used when combining different sources of qualitative and quantitative data, and it can be used to improve the reliability of the research [28]. In this research we designed our data collection methods to support methodological triangulation. The interview collected only qualitative data, and the survey collected both qualitative and quantitative data. Table below summarizes how our data sources answer to

research questions:

Table 3.1: Summary of how data sources answer to research questions

Method	RQ1	RQ2	RQ3	RQ4	RQ5
Literature review	X	X	X	X	X
Interviews	X	X	-	X	-
Survey	X	X	X	X	X

3.2 Literature review

In addition to the case study, the literature review was also used to answer all the research questions. Our literature review combines a systematic literature review(SLR) made by Shahin et al. [30] and our own literature research. The SLR used a well defined search word[1]. The articles of the SLR were selected following a predefined inclusion and exclusion criteria: Articles needed to be peer reviewed, more than 6 pages long and required to have empirical results. Editorials, position papers, keynotes, reviews, tutorial summaries, panel discussions and non-English were excluded. The articles were selected by applying inclusion and exclusion criteria while reading the abstract and conclusion of the articles. Articles were also searched by using snowballing technique: The referenced articles of selected articles was added to the literature review if they matched the criteria. The final result was 69 articles [30]. Some of the primary sources of SLR are cited in the literature review to provide more detailed information.

The literature research of this thesis was made using Google scholar. Search was made using the search word "Continuous Deployment" and only accepting studies from year 2017 onwards because these articles are so new they are not included in the SLR. The studies related to CD were included if they had some empirical research and provided useful information towards our research. The first 100 search hits sorted by relevance were analyzed. The selection was made by first selecting the articles which had title that suited the research. After this phase we had 22 articles. After that we read the abstract and conclusion of each article ,

[1](("continuous integration" OR "rapid integration" OR "fast integration" OR "quick integration" OR "frequent integration" OR "continuous delivery" OR "rapid delivery" OR "fast delivery" OR "continuous deployment" OR "rapid deployment" OR "fast deployment" OR "fast deployment" OR "quick deployment" OR "frequent deployment" OR "continuous release" OR "rapid release" OR "fast release" OR "quick release" OR "frequent release" OR "deployability" OR "continuous build" OR "rapid build" OR "fast build" OR "frequent build" OR "quick build") AND ("software" OR "information system" OR "information technology" OR "cloud*" OR "service engineering"))

and evaluated is the article useful towards our research. In this phase eight articles were selected. In addition we conducted a search where we used the same search word as in SLR and accepting articles 2017 onwards from Google scholar. The goal of this search was to evaluate, what results the search word of SLR would produce in new articles. We analysed the first 100 hits according to relevance. Out of the total of eight articles of our literature research only three articles were found.

The literature review, which includes the SLR and our literature research is used to answer all research questions. RQ3 and RQ5 take into account how often the problems and best practices happen. When answering these research questions, we focus on articles which have quantitative data about how often the problems or best practices happen. For example a percentage of respondents in a survey, or how many projects from total amount of projects in an organization state that the problem or best practice is occurring. This kind of data represents how often the problems and best practices occur in real-life scenarios.

3.3 Interviews

Runeson and Höst [28] highlighted the importance of interviews in case studies [28]. The motivation of interviews is to understand current benefits (RQ1), problems(RQ2) and best practises(RQ4) of CD in Eficode's projects. This allows for further analysis referring to problems and best practices found in the literature review. The goal of the interviews was to identify do the problems and best practices identified in literature review happen in real-life scenarios. The interviews contained questions which asked the opinion about the upcoming survey. The goal of these questions was to collect tips to plan the survey from experts that have long experience at both working at Eficode and with CD projects in general.

Runeson and Höst [28] described that in semi-structured interviews the questions are planned, but they are not specifically asked in the same order as listed. Semi-structured interviews allow for improvisation and exploration of the studied objects [28]. In this research semi-structured interviews were chosen to enable broad view of the subject and not to limit the interviewee. During this research 2 semi-structured interviews were conducted, one with Eficode's software team leader and one with Eficode's CTO. The justification for these selections was that both have a long background in software development and have seen Eficode's CD projects in a long time frame.

Runeson and Höst [28] encouraged to collect the interviews in audio format and to transcribe them before analyzing the data [28]. The interviews were held in Finnish as it was the native language of both interviewer and interviewees. In this research we recorded the audio of both interviews with the permission of the interviewees and transcribed the speech to written English.

3.4 Survey

Next we introduce the survey, which was the main data collection method of our case study. The survey is used to answer all our research questions. It answers, does a large crowd acknowledge the benefits, problems and best practices of CD. It also gives detailed data about how often these problems and best practices occur in real-life scenarios. We start by going through the methods that were used to plan the content of the survey, then we go through what characteristics the final survey included. We end the section by going through the whole process of planning the survey.

3.4.1 Iterative planning of the survey

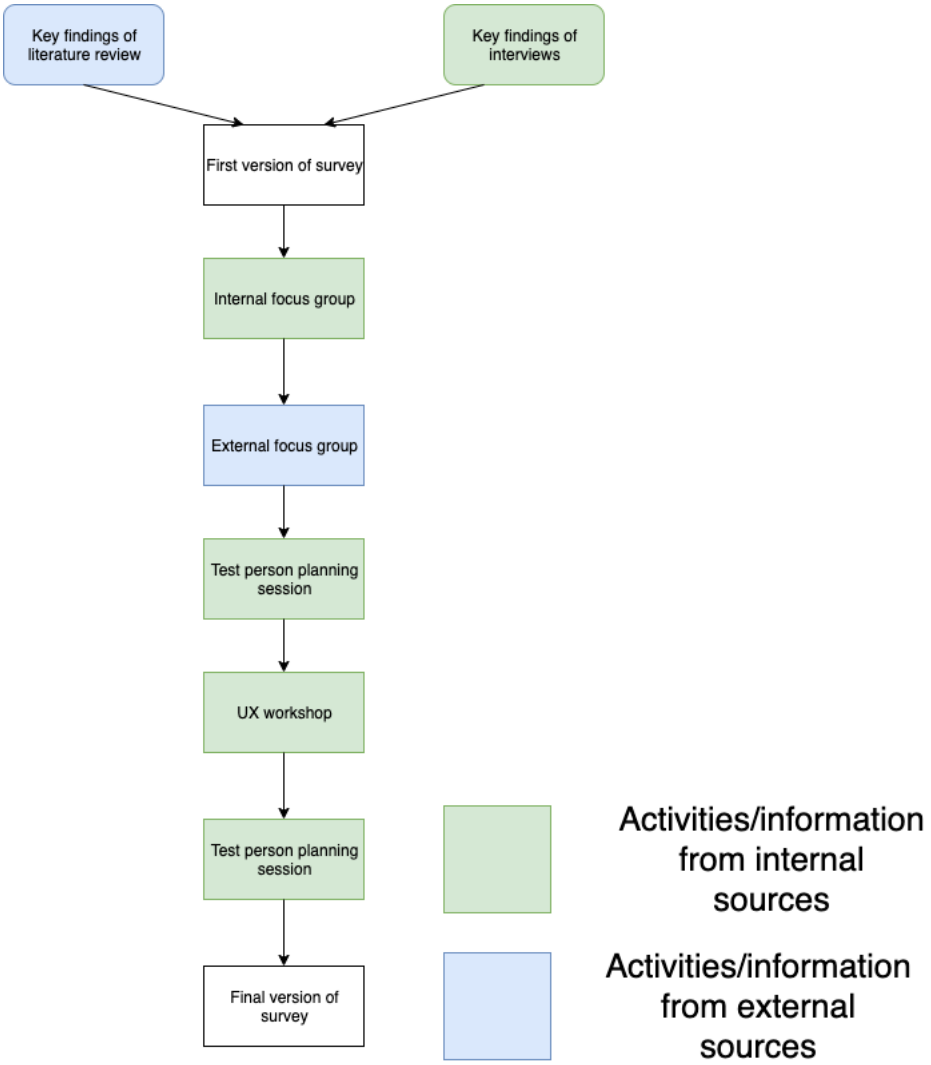

Figure 3.1: Phases in iterative planning of survey

Figure 3.1 represents the iterative planning of the survey, which included 7 steps. After each step the survey was redefined to fit better the research needs. Survey was designed in iterations to learn how the questions are understood by the participants. The iterative process started with focus groups, which are lighter ways of evaluating the survey. The activities became more demanding towards the end of the planning phase, meaning test person sessions and UX-workshop. The reason to use more lighter ways in the beginning was to gather feedback easier and faster for the first version of the survey, that were more easily noticed. In the end of

the planning it was beneficial to use more demanding activities to find the last modifications that made the survey more understandable.

Next we go in detail the survey planning methods. Kitchenham et al. [15] wrote guidelines for designing a survey. Using a small focus group for designing the survey is useful to gather feedback from the survey [15]. The strength of the focus group is that it reveals how the survey is answered by users, which would answer the survey. Focus groups produce a small amount of data that the actual survey would produce. It is also a fast and easy way of gathering data and feedback about the survey. The weakness of focus group is that it might not gather feedback on the weaknesses of the survey, when the respondents are focused on answering the survey. The weakness of the internal focus group is that the respondents can't answer the final version of the survey, as it may produce biased results.

Our survey planning process included two focus group sessions, one was held internally and one externally. The first focus group contained six persons. The session was held internally at the Eficode office. The second focus group contained four external software professionals, which were gathered from the network of researchers. The reason to use an external focus group was to keep the sample size of the original survey high and to gather new insights from outside of our case.

Test person planning sessions were used to gather straight-forward feedback about the usability of the research. In these sessions the survey was filled by a person in the same room as the researcher. Test person could ask questions and specifications from the researcher from unclear parts of the survey. The strength of the test person session was that the direct discussion about the features of the survey gave very detailed information on how the survey is understood. The weakness of the test person planning session was that it took longer and more effort as it had to be in a certain time slot where the test person and researcher physically attend.

Our UX-workshop focused on improving the usability of the survey. The UX-workshop was attended by senior content writer, usability expert and the researcher. Senior content writer offered help on writing questions in clear way, and usability expert gave feedback on overall usability of the survey. The strength of UX-workshop was that it provided great feedback from usability from non-technical experts.

3.4.2 Characteristics of the survey

Runeson and Höst [28] recommended to combine quantitative and qualitative data in case studies [28]. The survey contains a pair of closed questions based on Likert scale, and open question related to the closed question. This way respondents can answer their specific opinion on each section relating to CD related problem or best practice, and use their own words to describe the reasoning behind the answer.

Kitchencham et al. [16] mentioned, that if the survey is too long, the respondents might lose interest to answer with care to each question [16]. The weakness of using both quantitative and qualitative questions is that it makes the survey longer. Kitchencham et al. [16] recommended to order the questions in groups in order to compensate against the number of questions [16]. The questions were divided into sections, which all related to the same subject. Also all the questions in the same section followed the same pattern to make the structure of the survey more clear and compensate for the large amount of questions. Survey was designed using Google Forms, since it provides built-in data collection and analysis tools.

Survey was targeted at Eficode's software and DevOps team. This part of Eficode's organization provides great knowledge of CD in the context regarding our research problems. Eficode's Software and DevOps teams are involved in plenty of projects which use CD. Having such a large scale of expertise gives a great data point for survey.

The survey was launched on 26.4.2019 at the Eficode office. The purpose of the survey and thesis was explained during company-wide info session, and the survey was launched in internal Slack channels and email lists. Reminders to answer the survey was sent bi-weekly on Slack channels. Survey was closed on 6.6.2019.

Chapter 4

Results of literature review

This chapter will start by going through the content of our literature review. Then we will provide knowledge about why CD is a popular method in modern software development which makes it worth researching. We also determine the most common problems and best practices of CD found in literature review.

4.1 Content of literature review

We analyze and compare the results of systematic literature review (SLR) made by Shahin et al. [30] and our own literature research. The SLR was published 2017 and contains 69 papers, that were selected by following a predefined criteria. It thoroughly analyzes the tools, problems and best practices related to CD [30]. Our literature research includes 8 articles, which were not referenced in the SLR. We use both primary sources of SLR and our literature research for identifying benefits of continuous deployment, because benefits of CD was not one of the research questions in the SLR. When analyzing problems and best practices, our literature research is compared against the summarized results of the SLR. In addition we have searched for additional details from the primary sources of SLR. Table below summarizes articles from our literature research:

Table 4.1: Articles of our literature research

Title	Data type	Focus	Reference	Year
The top 10 adages of continuous deployment	Quantitative and Qualitative	Best practices	[25]	2017
Revisiting continuous deployment maturity: a two-year perspective	Quantitative and Qualitative	Best practices	[20]	2019
Continuous Delivery: Overcoming adoption challenges	Qualitative	Problems and best practices	[6]	2017
Microservices: Architecting for Continuous Delivery and DevOps	Qualitative	Best practices	[7]	2018
Building lean continuous integration and delivery pipelines by applying DevOps principles: a case study at Varidesk	Quantitative and Qualitative	Best practices	[12]	2018
Overcoming Challenges with Continuous Integration and Deployment Pipelines When Moving From Monolithic Apps to Microservices: An experience report from a small company	Quantitative and Qualitative	Best practices	[11]	2019
One size does not fit all: an empirical study of containerized continuous deployment workflows	Quantitative and Qualitative	Problems and best practices	[34]	2018
A case analysis of enabling continuous software deployment through knowledge management	Qualitative	Problems and best practices	[9]	2018

Next we will introduce the articles of our literature research. The article of Parnin et al. [25] is based on a survey relating to continuous deployment best practices, which was filled by 17 teams from nine companies. The article also represents results of discussions on the continuous deployment summit, which had participants from 10 software companies [25]. The study of Mäkinen et al. [20] also

focuses on best practices of CD. This case study focuses on Finnish software company called Solita, which has more than 600 employees. Solita's core competence is delivering projects as a service. Each project is tailored towards the customer's needs. The case study relies on a survey relating to best practices of CD [20].

The articles [6, 7] focus on a company named Paddy Power. Paddy Power relies heavily on increasing number of websites, mobile applications, trading and pricing systems, live-betting-data distribution systems and software used in betting shops. These applications are developed and maintained by the technology department, which employs approximately 500 employees. The adoption of DevOps and CD at Paddy Power began in 2012. Four years after the transformation, Paddy Power implemented CD for over 60 applications. The first study [6] focuses on problems and best practices of CD. The second study [7] goes through benefits of microservices in CD. Also the studies [11, 12] focus on microservices. The articles focus on a company called Varidesk, which offers a multitude of active workspace solutions for home and office spaces. Both articles focus on changing the architecture of the software of monolithic application to microservices [11, 12].

The study of Zhang et al. [34] focuses on the evolution of CD workflows caused by containerization. The study collected data from open source projects and from a survey, which was responded by over 150 developers [34]. The study of Colomo et al. [9] is a grounded theory study focusing on transformation to continuous deployment. The study focuses on DevOps team of 10 professionals. Data is collected from semi-structured interviews. The company is Meta4, which is a world leader in human capital management solutions [9].

4.2 Benefits of Continuous deployment

Next we introduce the different benefits identified from our literature research and the primary sources of the SLR. These benefits provide details why CD should be used in software development. The benefits can be reasoning for companies why CD should be invested into.

4.2.1 More frequent and faster releases to production

Many articles stated faster releases to the production environment as one benefit of CD [4, 5, 23, 25]. Chen [5] reports that the adaption of CD in Paddy Power increased the release frequency from less than six times a year to every week [5]. Neely and Stolt [23] investigated a transformation to CD in a company called Rally Software in their article. The company used to have an 8 week process to plan their next release, which was made in one day. This process had many flaws regarding deployment cycles: Implemented features had to wait in the worst scenario many

weeks to be deployed into production, or if the feature is not implemented to release day, it had to wait 8 weeks to be deployed. They transformed to using CD, so that they were able to release the software whenever they wished [23]. In addition faster releases cycles lead to faster customer feedback [5, 25].

In addition to the shorter release cycles, CD can also shorten the time used to actually release the software. Callanan and Spillane [4] had impressive results regarding to automation in their article. The time used for deployments decreased 86 percent and they increased their amount of releases 2.6 times from after transformation to CD [4].Also developers can enjoy when their code changes are going faster to production [25, 29].

4.2.2 Quicker fixing of defects

Continuous deployment makes fixing defects from software faster [5, 23, 25]. Neely and Stolt [23] described challenges in their old eight week periodic release process. The last week before release was used to find defects from the software. It was a hectic week, where all defects needed to be found and fixed , and the release was prepared. The last week was really expensive for the company. After their CD transformation they listed many benefits relating defects: Small code changes lead to less defects in production, and if defect was noticed, it was easier to track when the amount of changed lines of code is smaller [23].

According to Chen [5],the former release process of Paddy Power was not properly practised by developers, and it consisted of many manual, error prone activities. They also had to do intense bug searching before releasing the product. After transforming to CD, the amount of bugs reduced over 90 percent, and if bugs occurred, they were fixed in a couple of days [5]. In the summit of Parnin et al. [25] companies stated that frequent releases of CD make defect fixing easier. [25].

4.2.3 Increased productivity

Savor et al. [29] found quantitative results, that *CD does not inhibit productivity or quality when scaling the codebase by 50x and team's size by 20x.* This result was gained by observing version control changes and failure errors from 6 years of time [29]. This is a solid result which seems to guarantee that CD is the productive solution for large scale software development. Also other studies noticed the productivity of development teams when using CD: Neely and Stolt [23] noticed in their study that automated testing in CD pipeline lead to more features implemented in long term [23]. According to Parnin et al. [25], teams believed that they were more productive and had better overall collaboration when using CD [25].

In the semi-structured interviews of Leppänen et al. [19] to software companies it was noticed, that CD helped to streamline the process and eliminate manual

work [19]. Chen [5] mentioned in his article, that in Paddy Power creating the test environment might take even 3 weeks of developers work. When transformed to CD, the test environment was initialized automatically [5]. CD can help save developers time from manual activities that are repeated during the quality assurance of the software.

4.2.4 Decreased stress of developers

Using CD can make the developers more motivated, and less stressed [5, 25] Chen [5] mentioned that in Rally Software development teams stress was reduced when CD transformation made releases more frequent action [5]. CD can help developers be less afraid to do deployments into production. Neely and Stolt [23] reported one case, where the developer was first afraid, that the code change he makes goes straight to production. After the CD transformation the developers had better work life balance, because the developments happened more frequently, which is less stressful than having one big deployment full of risks less frequently [23].

4.3 Development related challenges

Next we go through different problems and best practices which are related to activities of development team. Except for one problem which was found from our literature research, the source of the problems was the SLR. We begin each section by introducing the problem using the results of SLR, and introduce the best practice which solve this problem. In some parts details are added from the primary sources of the SLR. Then we present the results found from our literature research relating to the problem and possible best practices.

4.3.1 Lack of automated tests

According to SLR of Shahin et al. [30], low test coverage is one of the challenges of CD. Many of the studied organisations were unable to automate all types of tests. The lack of tests was shown in the primary studies of the SLR, for example in interviews made by Olsson et al. [24] 3 out of 4 companies wished that they had better quality automated tests [24]. One of the suggestions for improving testing activities such as low test coverage in the SLR was to use Test Driven Development (TDD) [30].

Also our literature research identified lack of tests as a problem. In the survey of Parnin et al. [25], automated acceptance tests were not widely used among the companies: Little over 25% of the respondents used automated acceptance

tests "All of the time". The responses for unit tests were much higher: Almost 75% of the respondents used automated unit testing "All of the time". In the interviews respondents answered that lack of automated testing can affect the CD process negatively [25]. In the case study of Mäkinen et al. [20] three out of eight projects used automated user acceptance tests for quality assurance. Despite the low amount of automated user acceptance tests, all projects except one had unit tests [20]. According to Parnin et al. [25] one of the best practices to fight against this problem is to invest in automation: It is one of key elements to stay competitive and survive [25].

4.3.2 Lack of quality in automated tests

In the SLR Shahin et al. [30] stated unreliable tests as one of main reasons of poor test quality [30].The interviews of Leppänen et al. [19] noticed, that developers have problems automating automated acceptance testing [19]. Some articles reported unreliable tests, which fail only in some circumstances. The reason for failure can be caused by the order of execution between test suites or synchronization issues, which are hard to reproduce [13, 23].

Callanan and Spillane [4] mentioned that in Wotif their data in the staging environment did not match the type or volume as in production. This led to defects in production, because the issues were not noticed in automated testing [4]. The SLR didn't provide any direct solutions to this problem, but many of the primary articles did suggest best practices for improving the quality of test automation: Savor et al. [29] suggested, that automated testing should always be done against same environment as the product should be, with same kind of data as in production. Virtualization can help to setup test environments that are identical to the production environment [29]. Also Gneimer et al. [13] mentioned that establishing test environment management is critical to achieve functional CD pipeline [13].

There are many ways, how test environments can be tailored to match the product environment: In the research of Gmeiner et al. [13] one team was assigned to provide anonymous database dumps to support the test activities [13]. Production database dumps are good to use in the test environment, because then the software will be running with the same data set as in the production. Neely and Stolt [23] mentioned, that Rally Software made automated testing even higher traffic loads than the production environment to make sure that the application has possibility to scale up [23]. None of the articles from our literature research noticed unreliable tests as an problem or introduced best practices to improve the test quality.

4.3.3 Long build times

Benefits of frequent builds vanish when the build times become too long. Long build times are a common problem when using CD. Shahin et al. [30] brought up long test execution times as one of the reasons for poor test quality, because long execution times prevent the fast feedback loop of CI server. Another problem that causes long build times is the large amount of test cases. Virtualization can be used to run tests in parallel as a solution against long build times [30].

The primary sources of SLR also brought up long build times and solutions for it: Neely and Stolt [23] mentioned in their article, that in one point the test suite took nine hours to run, which means that the product can't be possibly deployed in less than nine hours. It is encouraged to run the tests in parallel as soon as long build times are noticed [23]. In addition to long test suites, lack of computing power can also lead to long build times. Adams et al. [1] mentioned that slow build times can be a consequence of an organization having only one CI-server for many teams. Slow build times can be handled by decreasing builds or improving the performance of CI-server [1].

Also studies in our literature research identified long build times as an problem [11, 12, 34].In the case study of Debroy et al. [12] the builds were queued and they couldn't been cancelled. This caused a huge bottleneck in the CD process. In a newer case study of Debroy et al. [11], the average build and release time of their CD pipeline was one hour and 18 minutes [11]. According to one respondent of interviews conducted by Zhang et al. [34], build times over 20 minutes were harmful when trying to solve why build is failing [34]. In the case study of Mäkinen et al. [20] none of the eight projects had parallel tests [20]. Microservice architecture can help to shorten build times: Debroy et al. [11] accomplished 330 times shorter build times when using containerized microservice architecture [11].

4.3.4 Lack of knowledge of developers

Shahin et al. [30] noticed that several studies reported a significant gap in the required skills when implementing CD. The deployment and test automation demand new technical and soft skills. Improving team qualification and expertise can solve this problem. Several organizations provided formal training and coaching for developers. The aim is to improve the team's qualification and bridge the skill gap between team members [30]. The lack of knowledge of developer can be shown as mistakes of developers reported in the primary sources of SLR: Neely and Stolt [23] described one case, where tests failed because of old data in test environment, then the test was removed in order to deploy, resulting in bugs in beta environment [23]. Also Savor et al. [29] mentioned, that in OANDA skipping test in order to release lead to fatal bug in production. New developers can be

assigned to certain positions to learn the essentials of the CD process: In OANDA new developers must work at the release engineering for several months in order to learn the CD practices [29]. None of the articles from our literature research stated that the lack of knowledge of developers is a problem in CD or discussed best practices relating building team competence.

4.3.5 Code reviews

In the SLR code reviews were not seen as a challenge. Two studies of the SLR mentioned that some tools might not suit for reviewing code. [30]. However the primary article of SLR by Claps et al. [8] highlighted maintaining the code reviews as one challenge of CD [8].

Using code reviews is a good practice for handling code changes between members of the development team. One article from our literature research had quantitative data on how often code reviews happen: According to the article of Parnin et al. [25], developers prefer that their code is reviewed by a teammate before releasing the software. Code reviews were used by all respondents, and little over 50% of the respondents used them all of the time [25].

4.3.6 Architecture

According to the SLR of Shahin et al. [30] several studies indicated that unstable architecture creates hurdles when transitioning towards CD. For example inappropriately handled dependencies between components can cause challenges when implementing CD. Optimal architecture should be designed in a way, that each component of the software can be deployed independently [30]. This approach refers to microservice architecture.

Articles from our literature research noticed that monolithic applications have numerous challenges [7, 25]. Parnin et al. [25] stated that it can be challenging to achieve high frequency deployments with applications which have monolithic architecture [25]. Our literature research also suggested microservice architecture when using CD. Chen [7] found out many benefits of microservice architecture. They enable more simpler deployments and the ability to have zero downtime when deploying software. It is also possible to scale an service, which has been discovered as an bottleneck. Changing technology stack for specific service and updating libraries was also mentioned. According to Parnin et al. [25], Netflix uses microservice architecture, where teams are responsible for developing stable API interfaces [25].

4.3.7 Database schema migrations

Database schema migrations are used to make changes in a live database. They are important in software development, since the database schema will always be changed during the development process. Shahin et al. [30] highlighted database schema changes as a challenge when implementing CD. The solution for this problem is flexible and modular architecture, because each part of the software can be released independently [30].

If the architecture is modular, the database schema changes should be executed automatically without harming the other parts of the software: Callanan and Spillane [4] went through database changes in their article. They suggested that database schema changes should be backward and forward compatible. They also mentioned that Wotif used software called LiquiBase to automate database migration during release [4]. Having backward and forward compatible migrations leave room to change the schema if the migration did not work as planned. None of the articles from our literature research noticed database schema migrations as a problem in CD.

4.3.8 Configuration management

Configuration management was noticed as a problem only in our literature research. According to Parnin et al. [25], modern configuration management tools allow configuration management to be scripted and orchestrated across all server assets. Not treating configuration management leads to a significant amount of production issues at scale. Despite modern tools, configuration management can cause difficult errors: For example Netflix makes 60 thousand changes daily and has no system for tracking and reviewing them [25]. Also Chen [6] noticed managing infrastructures as a challenge [6].

Provided solution for configuration management was to treat it like source code and automate all infrastructure related steps: In the summit of Parnin et al. [25] companies suggested that configuration management should be treated as code from the start of the project [25]. According to Chen [6] infrastructure provisioning should happen automatically [6].

4.3.9 Team structures

According to the SLR of Shahin et al. [30] distributed development teams are associated with a number of challenges, for example lack of visibility. Defining new roles in the team is important when adopting CD [30]. Also the developers responsibility of the released software was emphasized: Developers should be on

call when the software is released, so they can see how the change affected the behaviour of end users, and react quickly if defects occur [29, 30].

The primary sources of SLR contained articles in which release engineering and testing was both the responsibility of a separate team and articles in which it was the joint responsibility of the development team. Savor et al. [29] mentioned in their article, that Facebook and OANDA uses Release Engineering team to handle pipeline and releases. The pros of this approach is that there are professionals allocated to take care of the releases and make sure that the pipeline and the infrastructures are maintained properly [29]. According to Adams et al. [1] Netflix uses a method called "Roll Your Own Releases": Developers, who implemented features are responsible for deploying it to production [1].

Savor et al. [29] discussed in their research that the need to allocate a separate team for testing is currently a popular discussion topic. In Facebook developers are in charge of testing [29]. However there are projects, where QA and testing is dedicated to separate team: In the case study of Gmeiner et al. [13] project had quality assurance team which was dedicated for writing automated acceptance tests and making sure the software functions correctly after deployment [13].

Our literature research had results relating to autonomic teams. Chen [6] encouraged for autonomic teams when transferring to CD [6]. Parnin et al. [25] noticed that also autonomic teams also have challenges: For example how autonomous teams integrate with each other. The solution for this question was using microarchitectures: Each team is responsible for keeping the microservice which they develop stable in each release. Netflix has a good example of an autonomic team, as they also have operations and quality assurance roles are embedded in the development team [25].

4.4 Organizational challenges

Next we go through different problems and best practices which are related to activities of the organization. First we introduce the problem using the results of SLR, and introduce the best practice which solves this problem. Details can be added from the primary sources of the SLR. Then we discuss the results found from our literature research relating to the problem and possible best practices.

4.4.1 Continuous deployment is not always an option

In CD the push to version control triggers automatic release of the software. However, in some cases this approach is not possible or desired. The SLR of Shahin et al. [30] indicated that some customers or domains don't prefer CD, and there wasn't a solution for this problem. [30].

The primary sources of the SLR provided detail, which domains are not suitable for CD: In the interviews of Leppänen et al. [19] one company worked with automation control system, where the production had to put to stop for a day in order to update the system. Another difficult area for CD is mobile applications. One of the companies interviewed by Leppänen et al. mentioned that the review of the application store took a week, and after that the latest version of application could be released [19]. Also Adams et al. [1] pointed out the difficulties when deploying mobile applications: Even small bug fixes have to go through the application stores review, and this is why quality assurance is in a critical role when developing mobile applications [1]. In these circumstances CD can't be used for deploying the software to the production environment at all times.

The primary sources of SLR noticed, that in some cases the releases are periodic or they require an manual acceptance phase. Leppänen et al. [19] noticed in their research, that 4 out of 15 companies wished to keep the releases periodic, varying from one to two weeks. They found the CD process disruptive for end users. 3 out of 15 companies wished to perform continuous delivery of the software, but not straight to production [19]. Chen [5] mentioned the manual acceptance phase in his article: Sometimes features can be implemented in a few days, but the acceptance phase takes 4 days to grant permission to release product [5].

Articles from our literature research didn't refer to certain domains that cause problems when adopting CD. However, according to Mäkinen et al. [20] the domain makes huge differences, when considering the best practices of CD. For some domains particular process areas are not relevant [20].

4.4.2 Lack of support from organization

According to the SLR of Shahin et al. [30] lack of trust towards CD was recognized as an issue. Also the challenge of changing established organizational policies and cultures was raised. [30]. Promoting team mindset and defining new rules and policies when adopting CD in organization was suggested as an solution for this problem [30].

The primary sources of the SLR provided details, why organization doesn't support CD: Leppänen et al. [19] noticed in their interviews, that in many companies the culture did not support CD [19]. Gmeiner et al. [13] brought up the importance of understanding the trade offs when investing to CD. Management needs to understand that constructing a CD pipeline is time consuming, and it might take time until the investment starts to pay off. Management support is crucial to survive the time until the benefits of CD are shown [13]. Also Neely and Stolt [23] emphazised the importance of management buy-in, as CD process affects everybody who is involved in the project [23]. When management gives team the support to do CD, their potential rises and they are not fearing that they

are not trusted.

According to Savor et al. [29] the risk management should belong to developers. This leads to better productivity when developers don't need to ask permission for every deployment. They also observed that the support from management may affect the productivity in the development team. OANDA's board almost changed from CD back to waterfall methods because of one mistake. The importance of having CD experts in the board of company to guide transformation and the direction of decisions was emphasized [29].

From our literature research one article noticed the lack of customer organization support: Mäkinen et al. [20] reported, that some customer organizations did not allow to use best practices. For example some customers did not have habit for implementing automated acceptance tests [20].Our literature research brought up the importance of culture [9, 20, 25]. According to Mäkinen et al. [20], it is crucial to improve best practices and to avoid being stuck to old processes [20]. In the summit of Parnin et al. [25] almost every participant had a story of bringing the whole software down with accidental mistake of configuration changes. In these cases blameless culture is important to recover and learn from mistakes. All summit participants use retrospective to support reflection on production failures [25].

4.4.3 Development team missing the right tools

Lack of suitable tools and technologies was one of the key challenges of SLR made by Shahin et al. [30]. However the SLR didn't provide any solutions for this problem [30]. Setting up CD pipeline requires knowledge of how to select right tools and how to use them, which was also noticed in the primary sources of SLR: Olsson et al. [24] noticed that the variety of tools were harmful to development in one organization [24]. Learning new tools is not probably easy, if they change all the time. Adams et al. [1] noticed that organizations tend to change their infrastructure as code providers if the results don't please them [1].

The primary sources of the SLR provided few suggestions for this problem. Savor et al. [29] suggested that all developers should be trained regularly to be able to work with different tools when adapting CD [29]. Chen [5] mentioned in his article, that when Paddy Power started their CD transformation, they built a platform, which allowed them to create a CD-pipeline for each new application [5]. Using an already made pipeline allows the team to start development work faster, instead of configuring each step of the pipeline for scratch. Pipeline templates are handy for companies which are going to have multiple CD projects in future.

Also the studies from our literature research highlighted the challenge of tools [6, 9, 12, 34]. Colomo et al. [9] stated that availability of tools is a challenge when implementing CD [9]. Zhang et al. [34] stated that developers face trade-offs when choosing the CI-tool. There is a lack of best practices for choosing CI-tools when

considering developers with specific needs [34]. According to Debroy et al. [12] In the beginning of the project a lot of resources were used to find the tools and learn how to use them [12]. The summit of Parnin et al. [25] suggested a solution for tooling issues: Large organizations have a team, which is focusing on tools. This team enables the tools for the development teams [25].

4.4.4 Schedule pressures

According to SLR high quality applications, that are supposed to be frequently released to customers may cause some team members to face more stress and extra efforts. The SLR didn't provide any solution for this problem [30]. Our literature research also recognized schedule pressures as an problem [9]: Software companies suffer pressure from customers to reduce release times while ensuring high quality [9]. Therefore schedule pressures can be caused both internal or customer organization. Our literature research didn't provide a solution for this problem.

4.4.5 Customer environment

The SLR of Shahin et al. [30] brought up challenges of the customer environment when adopting CD. For example, lack of access to the customer environment disrupts the efficient development work. Involving the customer more to the CD process was suggested as a solution for this problem [30]. From the primary sources of the SLR Chen [5] mentioned, that when Paddy Power transformed to using CD, they had to have root accesses for servers, which took six months of negotiations. CD demands breaking the boundaries between different teams, which is necessary to solve this problem [5].From our literature research Mäkinen et al. [20] brought up a challenge, that the development team has not full access to the production environment [20]. However none of the articles from our literature research provided solutions for this problem.

4.5 Summary of results

In this chapter we summarize, how the literature review answers our research questions. RQ1, "What are the identified benefits of continuous deployment?" is answered by combining the findings of SLR and non-SLR articles, following benefits were found:

- Faster releases to production
- Quicker fixing of defects

- Increased productivity
- Decreased stress of developers

RQ2," What problems happen in continuous deployment?" was answered by comparing the results of SLR and non-SLR articles. Table 4.2 below summarizes the problems that were found:

Table 4.2: Summary of identified problems

Description	SLR references	Literature research references
Lack of automated testing	[29] [19] [8] [24] [23] [30]	[25] [20]
Lack of quality in automated tests	[8] [13] [23] [30]	-
Long build times stall the development	[19] [23] [30]	[11, 12, 34]
Architecture does not suit for CD	[30]	[7, 25]
Schedule pressures	[30]	[9]
Database schema migrations	[30]	-
Responsibilities are shared to separate teams	[30]	-
Lack of knowledge of developers	[23, 29, 30]	-
Configuration management	-	[6, 25]
Development team missing tools and access rights	[24] [30]	[6, 9, 12, 34]
Organization does not support CD processes	[19] [13] [30] [5]	-
Schedule pressures	[30]	[9]
Customer environment	[30] [5]	[20]

Relating RQ3, "How often do the problems happen in continuous deployment projects?", the SLR of Shahin et al. [30] did not have any quantitative results about how often the identified problems occurred in industry scenarios. However two articles from our literature research provided data relating to how often automated acceptance tests are used. In both studies the use of automated acceptance tests was quite low [20, 25]. In the survey made by Parnin et al. [25] only a little over 25% used automated acceptance all of the time. In the case study of Mäkinen et al. [20] three out of eight projects had automated acceptance tests. Thus lack of automated acceptance tests is a problem which happens often.

The table 4.5 below summarizes the best practices, which answer RQ4, "Which best practices improve the continuous deployment process?"

Table 4.3: Summary of identified best practices

Description	SLR references	Literature research references
Run automated tests against production like environment	[13, 23, 29]	-
Improve team expertise	[29, 30]	-
Digital code reviews	[8]	[25]
Microservice architecture	[30]	[7, 25]
Automated database schema migrations	[4]	-
Run automated test cases in parallel	[23, 30]	-
Upgrade CI-server to make build times faster	[1, 23]	-
Treat infrastructure as code	-	[25]
Automate infrastructure provisioning	-	[6]
Promote team mindset	[30]	-
Embrace culture	-	[20, 25, 30]
Define new team structures	[30]	-
Customer involvement	[30]	-
Support team for providing tools	-	[25]

Relating RQ5," How often are the best practices used in continuous deployment projects?", the SLR of Shahin et al. [30] did not have any quantitative results about how often the identified best practices occurred in industry scenarios. Also only one article of the literature research had data on how often best practices are being used: In the survey made by Parnin et al. [25] all of the respondents used code reviews, and little over 50% used them all of the time [25].

Chapter 5

Results of the case study

In this chapter we present the results of our case study, which consisted of an interview and a survey. We end the chapter by summarizing the results.

5.1 Interviews

In this section we present the results of two interviews we conducted. The first interviewee was Eficode's software team leader. He has a long experience in software development and working in upper management. The interview with the software team leader took 21 minutes. Both interviews were held in Finnish. The second interviewee was Eficode's CTO. He also has a long background in software development in general. He has also seen the emergence of CD from its beginning. The interview with the CTO took 1 hour. Next we go through both interviews by subjects that were discussed. The interviews contained mostly the same questions. However, we asked a bit more organizational questions from the software team leader, and more technical questions from CTO. The interview questions with the team leader and the CTO are listed in Appendix A and Appendix B, respectively. The questions ans answers are translated from Finnish to English.

5.1.1 Benefits of continuous deployment

Both interviewees agreed, that CD is the preferred way of doing software development. When asked about the benefits of CD, the team leader brought up the simplicity it creates:

> "No gurus are needed for installation. Less human errors and more repeatability. If installation to staging environment is successful, we can be pretty sure it works for product environment as well."
>
> Team leader

The CTO emphasized the reduction of the time needed for implementing and deploying a new idea.

> "The time between an idea and a release is essentially waste in the Lean way of thinking, whether it's development, quality assurance or preparing for the release. Continuous Deployment is a paradigm that helps building a culture where the implementation work can be released as fast as possible. This changes the way organizations approach new requirements and releasing."
>
> CTO

5.1.2 Problems related to continuous deployment

When asked about the most common problems when adopting CD, both interviewees brought up the lack of knowledge of the developers as a major challenge. The team leader mentioned that most of the errors are initialized by the mistakes of the developers, which are caused by schedule pressures:

> "Setting up the CD pipeline requires work. Once setup is done the pipeline can be trusted. Problems most often occur in practical work. For example when a developer encounters schedule pressures he might forget to do something or tries to take shortcuts which might later require improving."
>
> Team leader

In the CTO's opinion most of the problems related to CD are caused by organizational structures and organizational strategy that does not allow the time needed to build the automation. In his opinion the developers usually do not have enough experience or knowledge on the different components of CD, or the development priorities simply do not give enough time to concentrate on building a solid CD. Unfortunately the lack of automated testing and development pipeline will collect technical debt, which in turn, will make reaching CD even more difficult.

The interview of the Team leader contained more comments relating to organizational problems. He brought up the challenges caused by client organizations:

> "CD is indeed the best practise, but we have to be flexible if the client's processes don't support it for releasing to production. In these cases it is an advantage for us if we use continuous deployment until staging environment. Then we let customer decide the deployment to production. We try to teach the clients, but it's hard with some clients having either old practices deeply established or operations limited by regulations."
>
> Team leader

Usually the clients let Eficode's developers release the software with CD tools. However the software team leader mentioned one case where the client didn't allow Eficode's developers to access the production environment.

5.1.3 Best practices related to continuous deployment

Both interviewees introduced many best practices. The CTO stated few best practices, which are requirements for using CD: The first one was extensive test automation harness. Without this the complete CD process usually cannot be automated as introducing a new feature might break existing functionalities. Another best practice for CD is digital code reviews. The understanding and quality of the code grows as the review is done by a peer with a predefined review process. Additionally the peer should be responsible for the new code so that there is a shared responsibility of each change. As the reviewer is responsible for the new code, it usually leads to writing better code in the first place.

Both interviewees proposed best practices that could shorten long build times. Team leader's solution was to investigate regularly practices, e.g. how the test execution times could be minimized. According to team leader upgrading hardware might also be one option shortening the time test runs take. In CTO's opinion the best solution is a proper test case management and optimization. However running automated acceptance tests in parallel will also help decreasing the build times and maintain independence between the test cases.:

> "Running automated acceptance tests (written for example with Robot Framework) in parallel is a solution to prolonged build times, where upgrading hardware works only to a certain point."
>
> CTO

According to the team leader the team needs to have good knowledge of test automation, especially in automated acceptance (high level) tests. In addition it is important to know what tools to use. Starting the development with pipeline and test templates can help to speed up the process. The CTO emphasized that the team should be able to deploy the software by themselves. In addition the CTO pointed out that organization should support CD processes in order to gain benefits. He emphasized the importance of management buy-in:

> "The management of the organization should support building automation and CD as part of the development work. If the priorities are not set for sustainable development speed, it usually start collecting technical debt and eventually slows down the development significantly."
>
> -CTO

Neither of the interviewees thought the separate release engineering team is necessary. In the team leader's opinion this team structure sounded like an old fashioned separate operations team as opposed to a modern cross-functional DevOps team. The CTO said that while such team structure can be selected, the implications should be known in advance. The organization should select a suitable DevOps Topology to follow in their organization.

When asked about the need for a separate testing team, the team leader brought up how different domains can affect the needs of testing resources:

> "When working with crucial software the test coverage is high, a separate testing team can be good. When working with normal web development where human lives are not at risk, test coverage can be lower. In these environments the development team usually writes the automated tests. Developer has responsibility of writing quality code which works, not putting it to testers responsibility."
>
> Team leader

CTO answered the same question on a more general level:

> "A separate testing team is really hard to make efficient in projects that have long life cycles. A testing team can provide the know-how and support for creating the test cases, but usually it is better that the development team maintains the automated test cases as part of their development work."
>
> CTO

5.1.4 Summary

Interviews provided answers to research questions RQ1, RQ2 and RQ4. Both interviewees agreed that CD is the preferred way of developing software. The identified benefits of CD were decreased human errors, more repeatability and a shorter time required to release the software to production.

The identified problems were related to developer mistakes or lack of knowledge. The CTO stated the organizational structure and management strategy are usually the main reasons for the problems. Also the lack of knowledge with the developers is a contributing factor. Team leader highlighted the fact that most problems happen by mistakes of developers.

Interviewees had some differences regarding answers to best practices. While running automated acceptance tests parallel in pipeline was a common solution to slow build times, the team leader was more positive about hardware upgrades to solve long build times than the CTO. The CTO emphasized the importance of digital code reviews. The team leader brought up using pipeline template for new

projects. Both interviewees had similar opinions relating to team structures: A separate release engineering or testing teams are not practical in smaller development teams, but might make sense in a larger project. The team leader mentioned that in specific industries a separate testing team can be useful.

5.2 Survey

In this chapter we go through the results of a survey released for Eficode's software professionals. The survey was sent to 184 persons and it received total of 24 answers. The full length survey can be seen in Appendix C.

5.2.1 Background questions

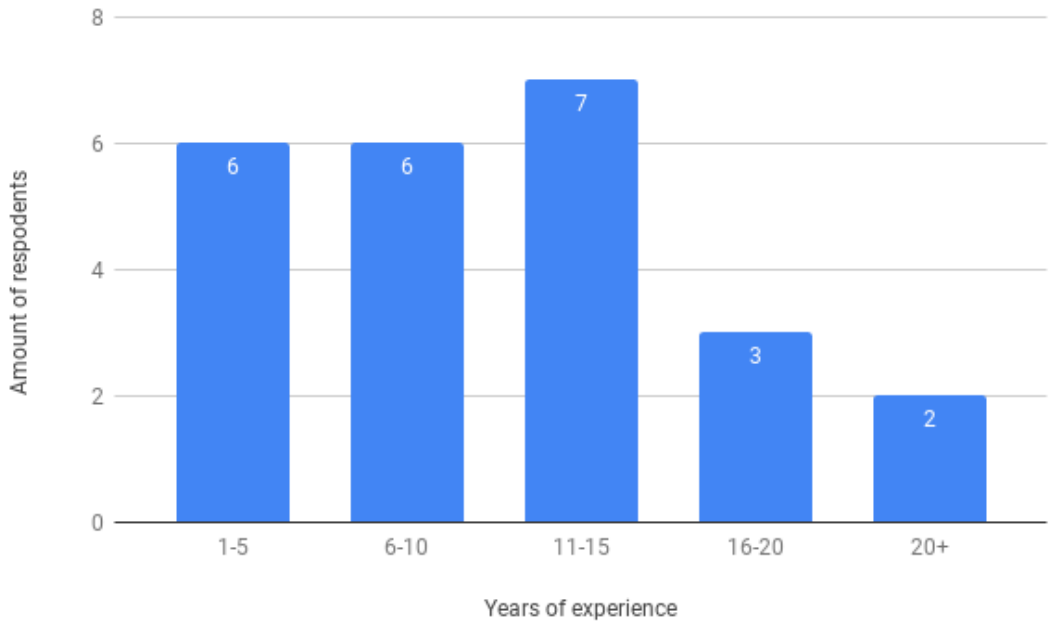

Figure 5.1: Experience represented as working years in software development (n=24)

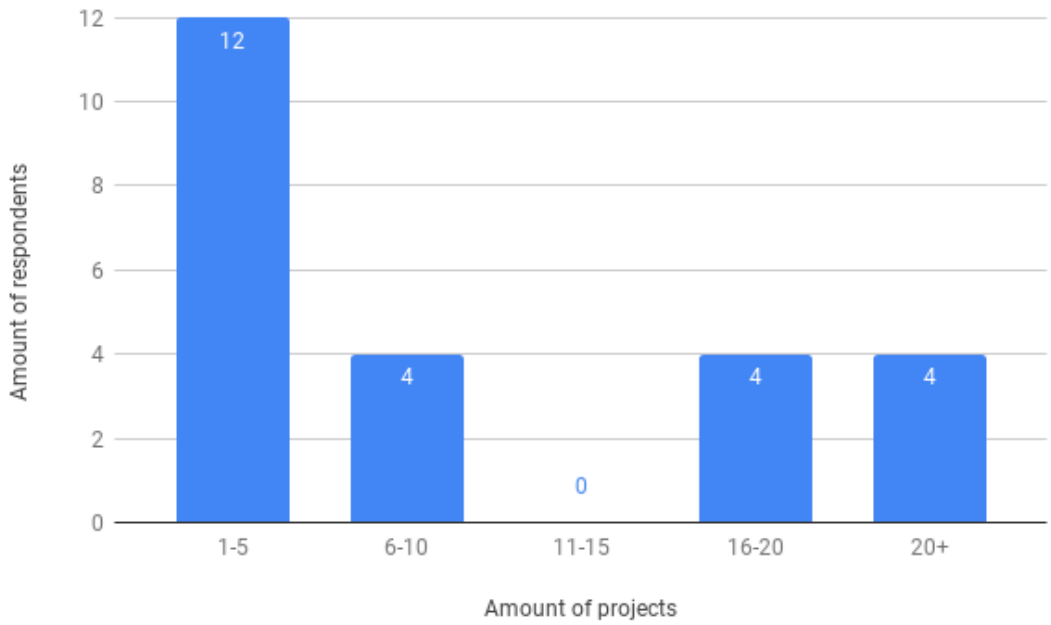

Figure 5.2: Experience represented as amount of CD projects participated in (n=24)

The experience of respondents are represented in figure 5.1 and figure 5.2. Our respondents had versatile career lengths and amount of projects which represent different amounts of experience working as a software professional.

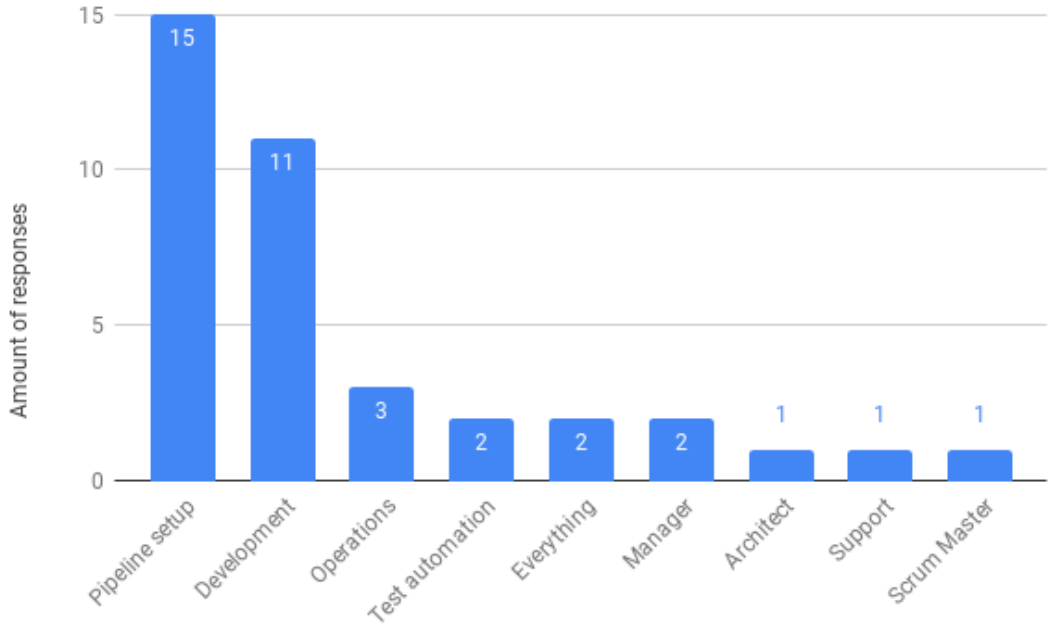

Figure 5.3: The responsibilities of the respondents in their CD projects (n=24). The number of responsibilities per respondent varied between one and three

Figure 5.3 represents the responsibilities the respondents had in the CD projects they worked in. The data is based on an open question, where the respondents could freely describe their responsibilities. Many respondents mentioned more than one responsibility. The most common answer was pipeline setup, which refers to making the pipeline functional. The second most common answers were development, which means programming.

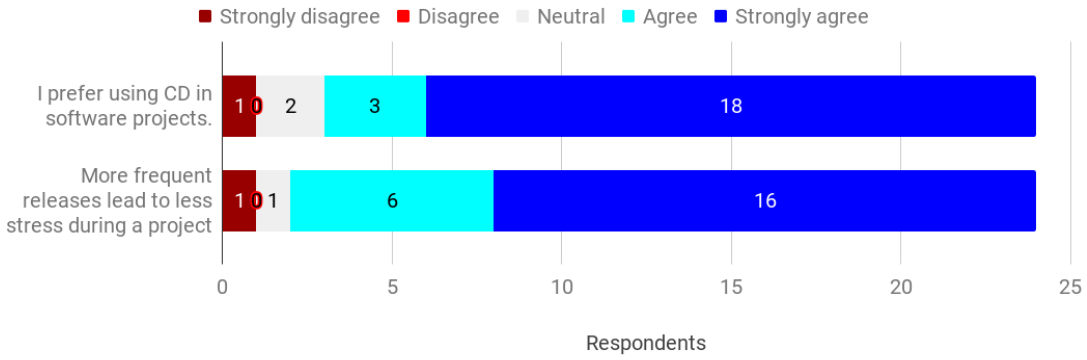

Figure 5.4: Respondents attitudes towards CD (n=24)

The answers related to attitudes towards CD are represented in figure 5.4. Both claims relate to opinions about the utility of CD. From the results it is clear that both statements were endorsed by respondents. For the first statement, "I prefer using CD in software projects" the most common justification was 'Faster feedback', which was identified from seven responses. There were also other justifications, but they received only a few answers. The following answer elaborates well why faster feedback may be an advantage:

> "The ease of deployment to a user acceptance test environment is very useful for developing and implementing fixes that require quick feedback from the customer and quick responses to that feedback."

For the second statement, "More frequent releases lead to less stress during project" Four respondents stated that more frequent releases make bug fixing easier. Six respondents motivated that frequent releases lead to smaller risk of failure, as in this answer:

> "More frequent = smaller batches. No stress about big deployments, Less of a chance for things to break when updating fewer features that are properly tested"

5.2.2 Problems related to continuous deployment

The problems and best practices were both examined by two variables: **Acknowledgement** and **prevalence**. The data of acknowledgement is based on answers of the mandatory Likert scale questions. The answer options are listed below:

- No experience
- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly agree

Choosing "No experience" indicates that the selected problem has not occurred in any work related projects. Choosing "Agree" means that the problem has happened in the work context and it is acknowledged as a problem. On the other

hand choosing "Disagree" means that the problem has happened in work context, but it is not causing major problems to development work.

The data of prevalence is gathered from voluntary open questions gathered from each problem. The questions are in form "How often this problem has occurred in your projects?". The answers represent the prevalence of the problems in real-life context. "No experience" answers from the mandatory part were included to indicating prevalence of that the problem has never occurred. The answers to prevalence questions were coded to options above:

- No experience
- Rarely
- Half of the time
- Often
- Very often
- Always

Each statement had an open question, which asked how the problem has been solved in the projects of the respondent. The answers from these questions were coded to possible solutions.

Table 5.1: Description of problems

Abbreviation in the figures	Description in survey
1) Database schema migrations	Based on my experience, database schema migrations lead to failing deployments
2) Lack of tests	Based on my experience, lack of automated test cases in the pipeline leads to bugs and problems in further development
3) Lack of quality in test cases	Based on my experience, automated acceptance tests (made with Robot Framework for example) in the pipeline are not revealing bugs
4) Schedule pressures	Based on my experience, schedule pressures lead to sloppiness and technical debt.
5) Long build times	Based on my experience, long build times stall the development
6) Manual steps	Based on my experience, manual steps(for example manual acceptance testing or doing database schema migrations manually) stall the development
7) CD related tools	Based on my experience, CD related tools don't support the development work

Table 5.1 above represents the description of the problems in the figures , and what was the longer format of the problem in the survey.

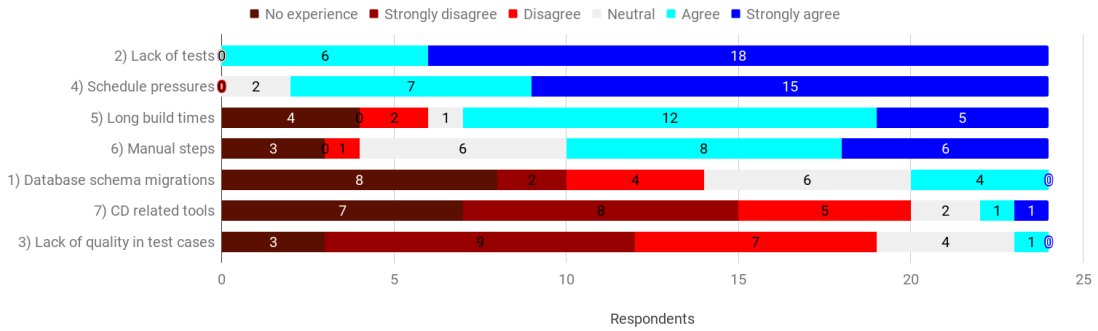

Figure 5.5: Acknowledgement of CD problems (n=24)

Figure 5.5 represents respondents acknowledgement to CD related problems. Problems can be clearly divided into two groups: Statements that were agreed by respondents, and statements that were disagreed or not experienced by respondents.

All respondents agreed to the statement 2, which related to lack of automated tests harming the development. Most common solution for this problem was to add more automated test, which was the answer of 10 respondents. Also four respondents suggested a straight forward fixing of the bug. One response brought up digital code reviews. Respondent suggested that the code change should not be possible to be merged to the master branch of version control unless all tests pass in CI-server. All respondents either agreed or answered neutrally to statement 4, which referred to the fact that schedule pressures leads to sloppiness and technical debt. There was no most common solution in the open question section. Adding resources to pay back technical debt received five answers, and limiting the scope of software received three answers. Two respondents stated that this problem has no solution.

Statement 5 relating to long build times received mostly ”Agree” and ”Strongly agree” answers. Most common solution was optimizing the pipeline, it collected nine responses. Optimizing included using cache, rewriting the pipeline to be more efficient and running tests or other parts in parallel. Improving hardware was the solution of three respondents. The fourth statement which received mostly positive answers was number 6, which stated that manual steps stall the development work. For this problem adding more automation was the solution of five respondents. Six respondents suggested redefining the process which includes a manual step. One respondent encouraged to document and improve the processes for the manual steps, and to look for bottlenecks and improve their throughput. Also two respondents suggested changing the architecture of the software.

Only a few respondents agreed to statements 1 , 3 and 7. For this reason

we don't go through the open questions about the solutions for these problems. When respondents were asked about other problems relating CD, one respondent brought up the monolithic architecture of software.

Next we analyze the prevalence of problems represented in figure 5.6 below. The question regarding prevalence was voluntary and not all answered it. The problems are ranked according to the highest median answer. If the problems have the same median, sum of "Often" , "Very often" and "In all projects" decides the order. The most prevalent problems were number 2, relating to lack of automated test cases and number 4 relating to schedule pressures leading to technical debt. Both received median of "Very often". All other problems received the median answer of "Rarely".

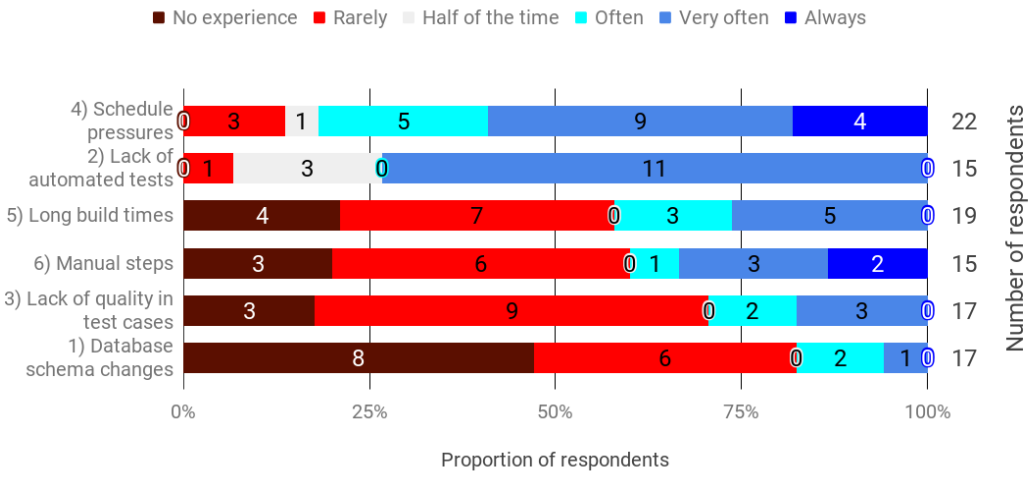

Figure 5.6: Prevalence of CD problems. (n=24), but the number of respondents varied from 15 to 22 per statements

5.2.3 Best practices related to continuous deployment

Table 5.2: Description of best practices

Abbreviation in the figures	Description in survey
1) Automated database schema migrations	Based on my experience, automated database schema migrations in the pipeline is a good practice
2) Run tests against database dump	Based on my experience, running automated acceptance testing (with Robot Framework for example) against respectfully handled production database dump in the pipeline is a good practice
3) Run tests in parallel	Based on my experience, running automated acceptance tests (with Robot Framework for example) parallel in the pipeline is a good practice
4) Pipeline template	Based on my experience, starting the project with complete pipeline template is a good practice.
5) Support team	Based on my experience, having a support team, which enables efficient tools and access rights for development team is a good practice
6) Digital code reviews	Based on my experience, allowing changes into master branch of version control only by digital code reviews (for example pull requests) accepted by another developer is a good practice

Table 5.2 represents the numbering, short format used in the figures, and the full format of how the best practices were asked in the survey.

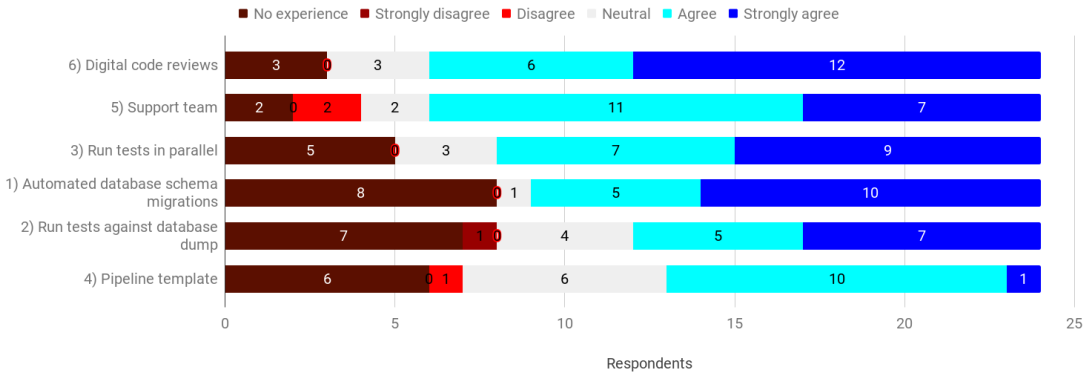

Figure 5.7: Acknowledgement of best practices in CD (n=24)

The best practices were examined in acknowledgement and prevalence in the same manner as problems described in previous section. The structure of the questions were similar. Also the data coding of the open question related to prevalence and calculating the response rate was made in similar fashion. Figure 5.7 represents the acknowledgement related to best practices of CD. It can be seen that all of the practices received mostly positive answers, as there are only 4 responses in options "Disagree" or "Strongly disagree".

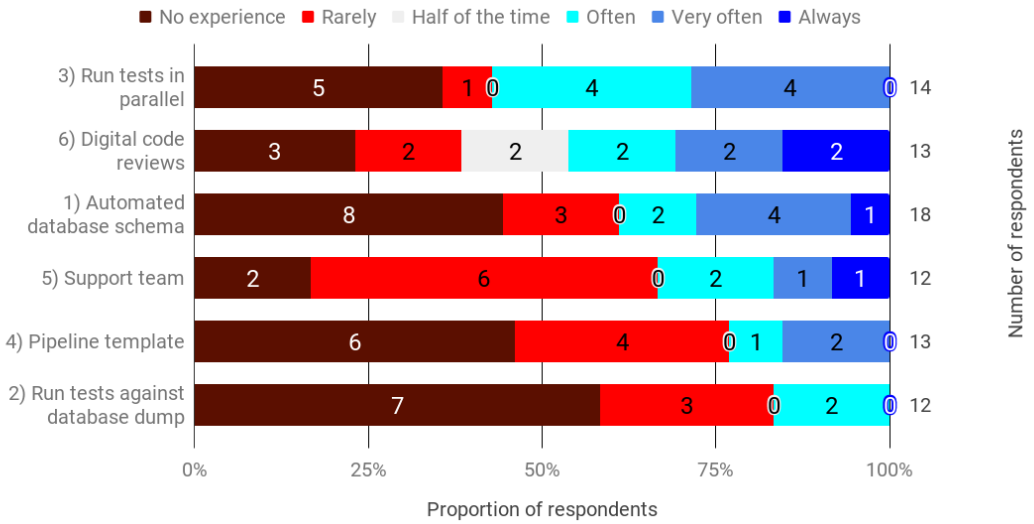

Figure 5.8: Prevalence of best practices in CD (n=24), but the number of respondents varied from 12 to 18.

Figure 5.8 above represents the prevalence of best practices. It can be seen that the best practices are not happening often in the projects. Practice number 3, running automated acceptance tests parallel, was the most prevalent practice.On the open question one respondent described that the tests should be independent:

> "If the tests are properly written, they should be completely independent. Therefore, the order in which you run them or executing in parallel does not affect the result."

Using digital code reviews (practice 6) was another common best practice. This citation from respondent gives examples why it is considered a best practice:

> "We have discovered many bugs when doing a peer review of the code before merging a merge request. It is also a good way for more people in the team to know and understand the code that is changed."

Some practices received answers why they were not used. Practice number 2, Running automated acceptance tests against respectfully handled database dumb was the least prevalent best practice. Three respondents mentioned that GDPR regulations relating handling personal information creates challenges to use this best practice. Three respondents stated that the reason for not having a separate support is lack of organizational resources.

5.2.4 Team structures of continuous deployment

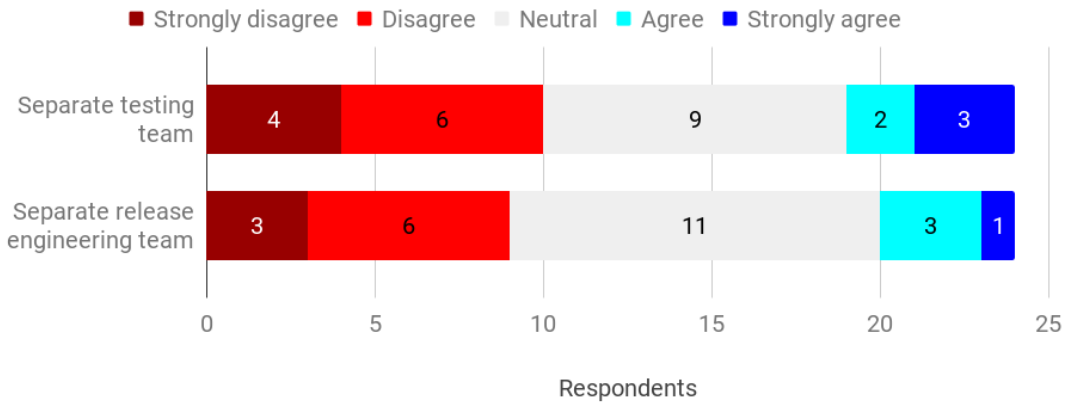

Figure 5.9: Answers related to the need for separate team structures (n=24)

Figure 5.9 represents the responses to questions about team structures. Both questions received answers from all options. In the qualitative answers section the positive sides of cross competence teams were brought up. On the release engineering question five respondents brought up that the responsibility of release engineering should be divided for the whole team, as in this answer:

> "Product team should have full responsibility over their product, including releasing it and running it in production."

Also in the separate testing team shared responsibility gathered most answers in the open question. This answer explains the motivation for cross competence teams:

> "Teams should be capable of handling almost any story independently. When creating specialised teams (like a test team) you are reducing this independence and increasing complexity. Cross competence teams are almost always the better choice."

5.2.5 Summary

Our survey indicated strongly that CD is the preferred way to do software development. Relating RQ1, the main reason for using CD in the survey was faster feedback. Respondents agreed that more frequent releases lead to less stress during the project, because it is easier to fix bugs, and because smaller releases have fewer risks.

The survey acknowledged the following problems related to CD, which answer to RQ2:

- Lack of automated test cases lead to bugs and problems in the future
- Schedule pressures lead to sloppiness and technical debt
- Long build times stall the development
- Manual steps stall the development

Relating to RQ3: The following problems received a median of "Very often":

- Lack of automated test cases lead to bugs and problems in future
- Schedule pressures lead to sloppiness and technical debt

To answer RQ4, most acknowledged best practices where:

- Running automated acceptance tests parallel in the pipeline
- Having a support team, which enables efficient tools and access rights for the development team
- Allowing changes into master branch of version control only by digital code reviews accepted by another developer

RQ5: The most common best practice was "Running automated acceptance tests parallel in pipeline", which had the median answer "Often". Digital code reviews had the median answer of "Half of the time". Questions related to using separate team structure received answers in both Disagree, Neutral, and Agree sections.

5.3 Summary of results

In this section we summarize the results of our case study and reflect how it answered to our research questions:

RQ1 asked "What are the benefits of continuous deployment?". All three data sources indicated that CD is the preferred way of working when doing software development. Literature review brought up less stress during development [5, 23, 29], faster releases to production [5, 23, 25] and increased productivity [4, 5, 19, 23, 29]. The interviews brought up the decreased human errors, and faster releases to production. The survey had answers which emphasized that the reason for using CD is faster feedback which is caused by more frequent releases. The survey also supported the point brought up in the literature review: More frequent releases lead to less stress during the project.

Table 5.3: Summary of the identified problems

Problem	Literature review	Interviews	Survey
Lack of automated test cases leads to problems in further development	X	X	X
Technical debt and schedule pressures lead to technical debt and sloppiness	X	X	X
Long build times stall the development	X	X	X
Database schema migrations lead to failing deployments	X		
Manual steps stall the development			X
CD related tools don't support the development work	X		
Automated acceptance tests are not revealing bugs	X	X	
Lack of knowledge of developers	X	X	
Customer environment	X	X	

Table 5.3 summarizes the results to RQ2 " Which problems happen in continuous deployment?". The lack of automated test cases, technical debt and long build times were identified by all three sources.

RQ3 "How often do the problems happen in continuous deployment projects?" is related to the prevalence of problems in the survey and to the results of literature review. The following practices happened most often in continuous deployment projects: Median answer for their prevalence was "Very often":

- Schedule pressures lead to technical debt and sloppiness
- Lack of automated test cases in the pipeline lead to bugs and problems in further development

Lack of automated tests was also recognized in thje literature review. Table 5.4 summarizes the best practices, which answer RQ4, "Which best practices improve the continuous deployment process":

Table 5.4: Summary of best practices

Best practices	Literature review	Interviews	Survey
Digital code reviews		X	X
Run automated acceptance tests in parallel	X	X	X
Use pipeline template	X	X	X
Support team	X	X	X
Automated database schema migrations		X	X
Run tests against database dump		X	X
Microservice architecture	X		
TDD	X		

RQ5 "How often are the best practices used in continuous deployment projects?" was taken into account in the case study in the prevalence of best practices in the survey. The most prevalent best practice was running automated acceptance tests in parallel, which had a median of "Often". Digital code reviews received the median of "Half of the time". Digital code reviews happened more regularly in the case study in the literature review. There more than half of the respondents used code reviews all of the time.

Chapter 6

Discussion

In this chapter we discuss the results of our case study. We start by discussing the benefits of continuous deployment. Then we discuss the identified problems, best practices, and their prevalence. They are both divided into development related and organizational related. Then we discuss the results relating to team structures. We end the chapter by evaluating the validity of our research.

6.1 Benefits of continuous deployment

Both literature review and case study indicated that CD is the preferable way of doing software development. One of the benefits of CD is more frequent releases to production. Bug in a popular system can have serious consequences. CD enables fast releases to the production environment, which can mean that bugs can be fixed within minutes once it is noticed. It is common that in software projects requirements change during as the product evolves. More frequent releases enable fast reactions to changing market needs. The development team can deploy features quickly and receive feedback from stakeholders.

Decreased stress of developers was also a benefit of CD. Deployments are made more often when using continuous deployment. This means that the developers become more experienced at making deployments as it becomes a daily task. They will not become afraid of hectic release days, where tens of code changes are deployed on the same day. Test automation might also be one reason for stress reduction. In CD the final decision to release the product is up to test automation, not single developer. This can lower the stress of the developer, as he/she is not in charge of the quality assurance phase: It is the whole team's responsibility to implement the test automation in a way that it covers all the business critical features.

This result signal that companies should invest resources on continuous deploy-

ment. CD is the preferred way of making software, as it enables faster changes to market needs and decreases the stress of developers. Achieving these benefits is worthwhile, as it enhances the quality of both development team and the product life cycle.

6.2 Acknowledgement and prevalence of problems

Next we discuss both the development and organizational related problems. We go through the impacts of the results and what could explain them.

6.2.1 Development related problems

The respondents of the survey acknowledged 4 problems out of 7. These 4 problems were also identified in the interviews. This means that some of the problems found in the literature review were not acknowledged as problems in our case company. This can be caused by a difference between the companies investigated in the case studies of literature review and our case company.

Our case study indicated that neglecting automated test cases and technical debt are problems in CD projects. These two problems are strongly linked to each other. Automated tests can cause problems in software development if they are neglected. Neglecting can mean that there are not enough tests, or the tests don't have high enough quality, which means that the tests don't detect bugs or they don't produce the same end result when executed multiple times, which means that the tests are not reliable enough. In chapter 2.9 we introduced technical debt in software development. Technical debt can also occur in testing, and neglecting automated tests can be thought of as one type of technical debt. The reason for technical debt can be schedule pressures, which was identified both in literature review and in the case study

In CD projects automated tests are the decision maker is software fit for deployment or not. In this case every business critical feature should have an automated test case to guarantee the feature works in the desired way. Software without tests can be thought of as technical debt, since the missing automated test is a thing that should be done, but it is left for later. Technical debt is mostly caused when trying to achieve results fast, and leaving the optimized solution to be done later. The root cause of technical debt might be linked to organizational causes: In the literature review we noticed that sometimes organizations don't support CD or understand what investments it requires: This might lead to under resourced project which leads to cutting corners and causing technical debt.

Our study identified long build times as one of problems in CD: This problem was acknowledged as a problem in literature review, interviews and the survey. The root cause for long build times can be simple: As the software grows, the amount of automated test cases grows. More tests in the pipeline means longer build times. Long build times decreases capability to build and deploy the software using CD. Long build times can lead developers to skip the tests, which can lead to bugs going to production as the tests are no longer detecting them. In a way long build times can be associated with technical debt: It can become an issue, which the team is aware of, but is not solved because the priority of the team might be doing new features and dealing with the long build times later. Stakeholders might not be pleasant, if the team uses most of their development time fixing long build times instead of developing features that make their product more provide more business value. The articles of literature didn't specify long builds as an form of technical debt.

Quality of automated tests was identified as an problem in the literature review, but not in the case study. The SLR brought up many cases, where the poor test quality caused problems in development work. However none of the articles from our literature research brought up this problem. It might be, that the sample from our literature research is so small, that none of the articles happened to have this problem. Another option would be that the test automation skills of development teams or the tools have improved during recent years, leading to the possibility of implementing high quality automated tests easier. Lack of quality in automated tests was not recognized as a problem in the survey of the case study. The reason might be that Eficode's consultancy is based on implementing high quality automated tests, which can be used in the CD pipeline. Database schema migrations was recognized as an problem in both literature review and the interviews, but not in the survey. The reason for this might also be that Eficode's professionals have a high routine to implement robust CD pipelines, which includes taking into account database schema migrations.

Literature review stated that software architecture can cause problems for the development team. This problem was not asked in the interviews or survey, because it was noticed from the literature review after the survey was closed. However one respondents of the survey answered that monolithic applications are a problem in CD. Both literature review and interviews agreed that lack of knowledge of developers is one problem of implementing CD. It is obvious that if the development team lacks the skills or knowledge how to develop software using CD is problematic. This problem can lead to other problems, which are already mentioned above.

Both the literature review and the survey of case study stated that lack of automated tests happen often in software projects. In literature review automated

unit tests were often used, but automated acceptance tests were not commonly used. The survey for the case organization didn't specify which test type are lacking, but the lack of automated tests was happening very often. These results mean that the most acknowledged problems were also most prevalent. From this knowledge can be concluded that projects are so busy that quality is compromised. Future studies could focus on the reasons why lack of tests happen in CD projects: Is it because the development team is missing the skills to implement them, or is it the schedule pressures caused by the organization?

6.2.2 Organizational problems

Adaptation of CD can be painful and unproductive if the development team is not capable of using the tools needed for CD. In the literature review having the right tools was seen as a problem when using CD. The tools can make the organization's IT-department run in circles: If the development team finds the tools unpleasant, the IT department needs to find new tools. This is repeated until the right tools are found, or the development needs to be started and the developers need to cope with tools that are not optimal. Also handling access rights was seen as a blocker for development work. Tools and access rights were not recognized as an problem in the survey. This might link to Eficode's consultancy is based on using tools to help customer projects being more successful. This includes teaching customer organization which tools are best for their situation and how to use them. Thus tools are not seen as a problem in CD.

According to literature review and the interviews of case study, lack of organization's support CD can be a major problem. This problem can heavily affect how resources are invested towards CD. Lack of resources can for example lead to a lack of suitable tools to use. Lack of organizational support can also lead to the developers not having enough support and trust to implement CD. Also, some organizations and industries don't fit for CD so well. This was also a problem that was recognized in the literature review and the interviews of case study. For these industries CD process needs to be tailored or replaced for some other choice. All of these organizational problems can be huge blockers for implementing CD efficiently. Organizations should take into account that development teams have enough knowledge to implement CD efficiently. Problems relating to customer environments were stated in both literature review and interviews.

Both literature review and our case study noticed that schedule pressures is a problem in CD. In the survey for the case organization lack of schedule pressures was happening very often. This problem makes the development team more pressured and vulnerable to mistakes. The schedule pressures can be caused by the organization, for example stakeholders asking for too much work without considering the realistic abilities of the development team. Organizational issues are

important to solve quickly, otherwise they will expand to development related problems. For example, if the development team is feeling schedule pressures it is likely that it will cause the team not having enough to implement automated test cases.

Future studies could focus on the root causes of organizational problems, for example which factors lead to schedule pressures in CD projects in an organization?Lavallee et al. [18] noticed in their case study, that upper management might force development team to assignments, which is not optimal for the quality of the software. This habit is endorsed by the fact that upper management is not usually up-to-date with the project status [18].

This result shows, how organizational decisions can affect software quality negatively. This is a notified problem in software development in general, and doesn't affect only CD projects: This article didn't specify was the project using CD or not. However, CD projects can encourage the development team to neglect quality if they are pressured by organization: For example CD can be used to deploy the software without test automated tests. In this way there is no quality assurance phase in the pipeline, but the software is deployed rapidly which might please the organization.

6.3 Acknowledgement and prevalence of best practices

Next we discuss both the development and organizational related best practices. We go through the impacts of the results and what could explain them.

6.3.1 Development related best practices

Digital code reviews were encouraged to keep enforcing the quality of the software in the interview and the survey. The literature review didn't bring up digital code review as an best practice, but that does not exclude that the CD projects would have used it. Digital code review is a good practice for making sure that the technical debt and untested code are not increasing in the first place. The respondents of the survey suggested adding automated tests and resources to payback technical debt as a solution to the problem of technical debt and lack of automated test cases. These solutions work for fixing the issue when harm is already done.

Running automated tests in parallel tests makes the test execution time shorter. This method was also introduced in literature review, interviews and the survey. In order to run test cases parallel, they need to be independent from each other. In this way it does not matter if the tests are executed at the same time. This requires

knowledge about test automation from the development team. If the team lacks this knowledge it is likely that the change process will be postponed until later. Another solution for long build times was to upgrade the hardware in the CI-server. This method was also covered by all three data sources. Upgrading hardware decreases the build times as the computing power of the CI-server increases.

These two solutions have different strengths and weaknesses. Upgrading hardware can make build times shorter temporarily, but does not solve the root cause. Let's imagine a situation where the automated acceptance tests phase takes 45 minutes in the pipeline. After upgrading the hardware the testing phase might be reduced to 20 minutes. However, running tests in parallel might reduce the testing time to about 15 minutes by running the tests in 3 threads. When this is combined to upgrading the hardware the build times could be under 10 minutes. Upgrading the hardware might be a short term solution to make the build times shorter, but running the tests in parallel can make the build times short and help them to sustain the short time easier. On the other hand, parallel tests require more knowledge and hands on work than upgrading hardware. It would be a good thing that the development team discusses these trade offs when facing long build times and figuring out possible solutions.

The literature review and few respondents of the survey suggested redesigning architecture as a solution to long build times. This solution solves the problem, by making each part of the software independent: Thus each part can be built, tested and deployed independently in the pipeline. This saves time as the development team can run pipeline tasks only parts that are wished. For example if the application has two parts: Frontend and Backend and the changes are only made to frontend. If the software is monolithic, the pipeline will also build the backend before testing frontend. If the software would be microservice architecture, the CI-server can test the frontend only because the backend has not changed.

The interviews and survey encouraged to run automated database schema migrations against database dump. This practice allows the development team to have similar data in tests as it is in production. The literature reviews did not recognize this best practice. The reason this might be, that this technique is so detailed, that it might not be covered in the CD studies. Same applies to automated database schema migrations: They were not recognized in any CD related studies. However this was recognized as best practice for both interviews and the survey.

Running automated test cases parallel in the pipeline received the highest median answer in the survey: "Often". However in the study of literature review none of the projects used parallel tests. This is a significant difference between our case organization and the organisation in the literature research. Even though the results of two surveys are not comprehensive to make assumptions, the sign

that some organisations don't use parallel tests at all indicates that this practice is not probably widely used in industry in general. Median for digital code reviews was "Half of the time". In one study of the literature review the code reviews were used more often: More than half of the respondents used digital code reviews all of the time. The prevalence of digital code reviews could be higher considering that it didn't receive any negative answers in the acknowledgement part of the survey. Future studies could focus on the reason why the prevalence of development related practices is not higher: If an organization identifies a set of best practices, it would be assumed that it would be used in almost all of the projects. However if the best practice is really technical, the development team might not have the skills to implement it in reasonable time.

6.3.2 Organizational best practices

Using already made pipeline template was identified best practice in literature review, interviews and survey. The pipeline template can fasten the adaption for the development team. It can have a significantly positive affect in the organization since many development teams can use the pipeline template. CD process can be much more efficient, if the culture and team spirit is supported by the organization.

Literature review and our case organization indicated that support team, which enables the development team access right and tools is a good practice. The responsibility of the support team is to enable tools and access rights for the development team so they can focus on development. This creates much more higher efficiency for development work, and also allows the support team to be able to investigate the usage of new tools, and to evaluate the usefulness of them. As a new project starts, the team should discuss what tools to use, and who are the persons that grant the access rights for these tools and maintain them.

The literature review encouraged to involve customers more in the CD process. However this aspect was not taken into account in the interviews or survey. Promoting team mindset and embracing supporting culture was another best practice, which was brought up in the literature review, but not taken into account in the interviews or the survey.

Organizational best practices are important, because they enable the development team to work more efficiently, and to fight against the problems that they are facing. The data of this case study suggested that organizational best practices are identified, but not widely used. Support team was one of the most acknowledged best practice, but it received a median of "Rarely". The reason for not having a support team was lack of resources. Another potential reason for not following acknowledged best practices is lack of expertise. Future studies could focus more deeply into why organizations are not capable of using these best practices.

6.4 Team structures

Team structures of software development is a hot topic to discuss, as agile mindset promotes cross-competence teams that are able to develop features from the requirements to the deployment. However, in some industries, for example healthcare more thorough testing is required and thus a separate testing team might be beneficial. The preference of team dynamics is domain and team specific issue, where there is not a single answer.

The literature review identified separated teams as a challenge. The solution for this problem was to redefine team structures that they suit development needs. The case study did not indicate clear sign of a certain team structure being the most productive for CD projects. This might link that every project is unique and the environment behind it affects what kind of team structures fit the project best. For example the domain, team size, experience might affect the team structure. Thus in new CD projects the team structure needs to be discussed with the development team and stakeholders by taking account following variables:

- Domain regulations
- Organizational boundaries
- Team size
- Experience of the team
- Stakeholder needs

For example, if the domain is a healthcare system for government projects, that affects people's lives. The regulation might be that every feature is acceptance tested by a named person. This can set organizational boundaries, that the team might need to name that are going to implement the testing. This needs to be taken into account when designing the team size and how experienced the developers should be. Separate testers crucially affect the development process of this team. Another example could be a company, which is developing a MVP for B2C product and trying to achieve a working prototype as fast as possible to receive feedback from end users. In this case cross-competence team is probably the most productive solution to produce software fast, which quality is high enough to produce the working solution to end users.

6.5 Validity of research

Runeson and Höst [28] discussed that the validity of a case study denotes the trustworthiness of the result, to what extent the result is true and not biased by

the researcher's opinion. Construct validity reflects to what extent the operational measures that are studied really represent what the researcher has in mind and what is investigated according to the research questions. The construct validity was taken into account by planning the questions on interviews and the survey carefully. However some of the questions in the survey was misunderstood: Many respondents understood the selection of "No experience" choice poorly: They stated that they agreed /disagreed with the problem, and then wrote in qualitative part that the problem or practice has never occurred for them in practice.

This research has taken into account both internal and external validity. Internal validity is concerned when causal relations are examined. When investigating whether one factor affects the examined factor there is a risk that the examined factor is also affected by a third factor [28]. Internal validity is taken into account in this research by planning the survey to have open questions, which allow the respondents to share their opinions on any factors which affect the statement. External validity is concerned to what extent it is possible to generalize the findings, and to what extent are of interest to other people investigated case [28]. The results are interesting to other software consultancy companies, because they enable them to capitalize the benefits of CD. This can boost productivity of software development and save costs.

Reliability is concerned with to what extent the data and the analysis are dependent on the specific researcher. Threats to this aspect of validity is, for example, if it is not clear how to code collected data or if questionnaires or interview questionnaires are unclear [28]. The survey did not receive as many answers as was planned. It received 24 answers from an organization of over 150 software developers / DevOps consultants. Also the response rates for prevalence of problems and best practices was relatively low. Also some of the respondents stated that the questions were unclear in their answers. The quality of the interview questions affects the reliability of the research negatively. Our literature review contained an SLR and our own literature research, which contained 8 articles. This literature research is not as detailed as the SLR which was cited. More thorough literature research might provide more accurate results. Also more quantitative results relating the prevalence of problems and best practices are needed.

Chapter 7

Conclusions

This thesis has found results, which indicate that CD has a number of benefits, which makes it a practical process for software development and worth researching. Faster releases to the production environment allow faster feedback and makes fixing defects faster. Using continuous deployment leads to more productive and less stressed developers, which should be the target for all software companies.

From the literature review and our case study we identified many problems, which harm the continuous deployment process. Schedule pressures and lack of automated tests were problems that happened most often in our case organization and literature review, which make them valuable for future research. Future studies could investigate the root cause of lack of automated tests and schedule pressures in continuous deployment projects.

Our study investigated best practices, which solve the identified problems and enable more efficient use of continuous deployment. Some of the best practices answered to identified problems: Running automated tests in parallel is a solution to long build times. Some of the best practices enhance the overall quality of the development work: For example digital code review helps the team to ensure the quality of rapid code changes.

However none of the best practices were highly used in our case organization or literature review. Future researches could investigate why these best practices are not frequently used. Does this problem link to the schedule pressures that was identified in the problems: Are organisations prioritizing releasing software quickly for customers instead of focusing on using the best practices to ensure quality?

Bibliography

[1] Adams, B., and McIntosh, S. Modern release engineering in a nutshell – why researchers should care. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (March 2016), vol. 5, pp. 78–90.

[2] Bacchelli, A., and Bird, C. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering* (2013), IEEE Press, pp. 712–721.

[3] Balaji, S., and Murugaiyan, M. S. Waterfall vs. v-model vs. agile: A comparative study on sdlc. *International Journal of Information Technology and Business Management 2*, 1 (2012), 26–30.

[4] Callanan, M., and Spillane, A. Devops: Making it easy to do the right thing. *IEEE Software 33*, 3 (May 2016), 53–59.

[5] Chen, L. Continuous delivery: Huge benefits, but challenges too. *IEEE Software 32*, 2 (Mar 2015), 50–54.

[6] Chen, L. Continuous delivery: overcoming adoption challenges. *Journal of Systems and Software 128* (2017), 72–86.

[7] Chen, L. Microservices: architecting for continuous delivery and devops. In *2018 IEEE International Conference on Software Architecture (ICSA)* (2018), IEEE, pp. 39–397.

[8] Claps, G. G., Svensson, R. B., and Aurum, A. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology 57* (2015), 21–31.

[9] Colomo-Palacios, R., Fernandes, E., Soto-Acosta, P., and Larrucea, X. A case analysis of enabling continuous software deployment through knowledge management. *International Journal of Information Management 40* (2018), 186–189.

[10] Cunningham, W. The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger 4*, 2 (1992), 29–30.

[11] Debroy, V., and Miller, S. Overcoming challenges with continuous integration and deployment pipelines when moving from monolithic apps to microservices: An experience report from a small company. *IEEE Software* (2019).

[12] Debroy, V., Miller, S., and Brimble, L. Building lean continuous integration and delivery pipelines by applying devops principles: a case study at varidesk. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2018), ACM, pp. 851–856.

[13] Gmeiner, J., Ramler, R., and Haslinger, J. Automated testing in the continuous delivery pipeline: A case study of an online company. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (April 2015), pp. 1–6.

[14] Haugset, B., and Hanssen, G. K. Automated acceptance testing: A literature review and an industrial case study. In *Agile 2008 Conference* (2008), IEEE, pp. 27–38.

[15] Kitchenham, B., and Pfleeger, S. L. Principles of survey research part 4: questionnaire evaluation. *ACM SIGSOFT Software Engineering Notes 27*, 3 (2002), 20–23.

[16] Kitchenham, B. A., and Pfleeger, S. L. Principles of survey research: part 3: constructing a survey instrument. *ACM SIGSOFT Software Engineering Notes 27*, 2 (2002), 20–24.

[17] Kruchten, P., Nord, R. L., and Ozkaya, I. Technical debt: From metaphor to theory and practice. *IEEE Software 29*, 6 (Nov 2012), 18–21.

[18] Lavallée, M., and Robillard, P. N. Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (2015), vol. 1, IEEE, pp. 677–687.

[19] Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V.-P., Itkonen, J., Mäntylä, M. V., and Männistö, T. The highways and country roads to continuous deployment. *Ieee software 32*, 2 (2015), 64–72.

[20] Mäkinen, S., Lehtonen, T., Kilamo, T., Puonti, M., Mikkonen, T., and Männistö, T. Revisiting continuous deployment maturity: a two-year perspective. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (2019), ACM, pp. 1810–1817.

[21] Mens, T., and Tourwe, T. A survey of software refactoring. *IEEE Transactions on Software Engineering 30*, 2 (Feb 2004), 126–139.

[22] Meyer, M. Continuous integration and its tools. *IEEE Software 31*, 3 (May 2014), 14–16.

[23] Neely, S., and Stolt, S. Continuous delivery? easy! just change everything (well, maybe it is not that easy). In *2013 Agile Conference* (Aug 2013), pp. 121–128.

[24] Olsson, H. H., Alahyari, H., and Bosch, J. Climbing the "stairway to heaven" – a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications* (Sep. 2012), pp. 392–399.

[25] Parnin, C., Helms, E., Atlee, C., Boughton, H., Ghattas, M., Glover, A., Holman, J., Micco, J., Murphy, B., Savor, T., Stumm, M., Whitaker, S., and Williams, L. The top 10 adages in continuous deployment. *IEEE Software 34*, 3 (May 2017), 86–95.

[26] Polo, M., Reales, P., Piattini, M., and Ebert, C. Test automation. *IEEE software 30*, 1 (2013), 84–89.

[27] Rogers, R. O. Acceptance testing vs. unit testing: A developer's perspective. In *Conference on Extreme Programming and Agile Methods* (2004), Springer, pp. 22–31.

[28] Runeson, P., and Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering 14*, 2 (2009), 131.

[29] Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., and Stumm, M. Continuous deployment at facebook and oanda. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)* (May 2016), pp. 21–30.

[30] Shahin, M., Babar, M. A., and Zhu, L. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access 5* (2017), 3909–3943.

[31] Spinellis, D. Version control systems. *IEEE Software 22*, 5 (Sep. 2005), 108–109.

[32] Stray, V., Moe, N. B., and Hoda, R. Autonomous agile teams: Challenges and future directions for research. In *Proceedings of the 19th International Conference on Agile Software Development: Companion* (2018), ACM, p. 16.

[33] Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., and Gil, S. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)* (2015), IEEE, pp. 583–590.

[34] Zhang, Y., Vasilescu, B., Wang, H., and Filkov, V. One size does not fit all: an empirical study of containerized continuous deployment workflows. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2018), ACM, pp. 295–306.

Appendix A: Interview questions for team leader

Common questions relating to CD

1. What is your background related to CD?
2. What are the benefits of CD?
3. What role does CD play in Eficode's software projects?
4. Do you consider CD a necessity in software projects?
5. Can you always use CD in software projects?
6. What are some things the team needs to be aware of before CD is useful?

Problems related to CD

1. What are the most common problems you have faced in software projects related to CD?
2. What is the solution to these problems?
3. Do automated tests often cause problems in CD?
4. Do long build times often cause problems in CD?
5. Do client organizations give enough support in CD projects?

Team structures

1. What opinions do you have relating to a separate testing team?
2. What opinions do you have relating to a separate release engineering team?
3. Any other opinions relating to team structures in CD?

Survey for developers of Eficode

1. What you would want to ask from Eficode's developer relating to best practices of CD?

Appendix B: Interview questions for CTO

Common questions relating to CD

1. What is your background related to CD?
2. What are the benefits of CD?
3. What role does CD play in Eficode's software projects?
4. Do you consider CD a necessity in software projects?
5. CD is not always an option, sometimes we use Continuous Delivery, what does it mean for developers? (branching in version control, waiting for acceptance phase)
6. What are some things the team needs to be aware of before CD is useful?
7. Using tools in CD, which problems or benefits can occur?
8. Are automated acceptance tests against staging environment a requirement for CD?
9. Are automated database schema migrations a requirement for CD?

Problems related to CD

1. What are the most common problems you have faced in software projects related to CD?
2. What is the solution to these problems?
3. Do automated tests often cause problems in CD? How is this problem fixed?
4. Do long build times often cause problems in CD? How is this problem fixed? Upgrading hardware, optimizing tests?
5. How does technical debt and schedule pressures show in CD projects?

Team structures and best practices

1. Which kind of developers are preferred when using CD?
2. What opinions do you have relating to a separate testing team?
3. What opinions do you have relating to a separate release engineering team?
4. Any other opinions relating to team structures in CD?
5. Opinions about pull requests when using CD?

Appendix C: Survey

Continuous Deployment practices survey

This survey focuses on Continuous Deployment and Continuous Delivery practices. Both practices are referred to as 'CD' in the survey. The goal of the survey is to collect CD related problems and best practices from work experience.

Thank you for contributing!

* Required

Background questions

CD = Continuous Delivery/Continuous Deployment, where the deployment happens through the pipeline automatically or manually

1. **How many years of work experience do you have in software development?** *

2. **In how many work related CD projects (1 project consists of 1 pipeline) have you worked on?** *

3. **What responsibilities did you have in the CD projects you worked on?** *

Questions related to attitudes towards CD

CD = Continuous Delivery/Continuous Deployment, where the deployment happens through the pipeline automatically or manually

4. **I prefer using CD in software projects.** *

   Mark only one oval.

   - ◯ Strongly disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly agree

Appendix C: Survey

5. **Which personal experiences lead to this answer?**

6. **More frequent releases lead to less stress during a project** *
   Mark only one oval.
   - ◯ Strongly disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly agree

7. **Which personal experiences lead to this answer?**

8. **In what projects you wouldn't use CD, and why?**

Problems related to CD

CD = Continuous Delivery/Continuous Deployment, where the deployment happens through the pipeline automatically or manually

9. **1. a) Based on my experience, database schema migrations lead to failing deployments** *
   Mark only one oval.
   - ◯ No experience
   - ◯ Strongly disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly agree

Appendix C: Survey

10. **1. b) How often has failing deployments related to database schema migrations happened in the projects you have been working on?**

11. **1. c) How has failing deployments related to database schema migrations been solved in the projects you have been working on?**

12. **1. d) If you wish, explain your answers to questions 1 a-c**

13. **2. a) Based on my experience lack of automated test cases in the pipeline leads to bugs and problems in further development** *
    Mark only one oval.
    - ◯ No experience
    - ◯ Strongly disagree
    - ◯ Disagree
    - ◯ Neutral
    - ◯ Agree
    - ◯ Strongly agree

14. **2. b) How often are bugs and problems caused by lack of automated test cases in the pipeline in the projects you have been working on?**

15. **2. c) How has the problems and bugs caused by lack of automated test cases in the pipeline been solved in the projects you have been working on?**

Appendix C: Survey

16. **2. d) If you wish, explain your answers to questions 2 a-c**

17. **3. a) Based on my experience automated acceptance tests (made with Robot Framework for example) in the pipeline are not revealing bugs** *
Mark only one oval.
- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

18. **3. b) How often have automated acceptance test cases not identified bugs happened in the projects you have been working on?**

19. **3. c) How has the problem of automated acceptance test cases not identifying bugs been solved in the projects you have been working on?**

20. **3. d) If you wish, explain your answers to questions 3 a-c**

Appendix C: Survey

21. **4. a) Based on my experience, schedule pressures lead to sloppiness and technical debt** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

22. **4. b) How often has schedule pressures caused sloppiness and technical debt in the projects you have been working on?**

_____

23. **4. c) How has sloppiness and technical debt caused by schedule pressures been solved in the projects you have been working on?**

_____

_____

_____

_____

_____

24. **4. d) If you wish, explain your answers to questions 4 a-c**

_____

_____

_____

_____

_____

25. **5. a) Based on my experience, long build times stall the development** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

26. **5. b) How often development has been stalled by long build times in the projects you have been working on?**

_____

Appendix C: Survey

27. **5. c) How has the stalled development caused by long build times been solved in the projects you have been working on?**

28. **5. d) If you wish, explain your answers to questions 5 a-c**

29. **6. a) Based on my experience, manual steps (for example manual acceptance testing or doing database schema migrations manually) stall the development** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

30. **6. b) How often development is stalled by manual steps in the projects you have been working on?**

31. **6. c) How has the stalled development caused by manual steps been solved in the projects you have been working on?**

32. **6. d) If you wish, explain your answers to questions 6 a-c**

Appendix C: Survey

33. **7. a) Based on my experience, CD related tools don't support the development work** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

34. **7. b) How often has the CD related tools not supported the development work in the projects you have been working on?**

35. **7. c) How has the problem of CD related tools not supporting development work been solved in the projects you have been working on?**

36. **7. d) If you wish, explain your answers to questions 7 a-c**

37. **8. Have you faced any other problems which are not described in questions 1-7?**

Best practices related to CD

CD = Continuous Delivery/Continuous Deployment, where the deployment happens through the pipeline automatically or manually

Appendix C: Survey

38. **1. a) Based on my experience, automated database schema migrations in the pipeline is a good practice** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

39. **1. b) How often database schema migrations are automated in the pipeline in the projects you have been working on?**

_____

40. **1. c) If you wish, explain your answers to questions 1 a & b**

_____

_____

_____

_____

_____

41. **2. a) Based on my experience, running automated acceptance testing (with Robot Framework for example) against respectfully handled production database dump in the pipeline is a good practice** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

42. **2. b) How often automated acceptance testing are run against respectfully handled production database dump in the pipeline in the projects you have been working on?**

_____

43. **2. c) If you wish, explain your answers to questions 2 a & b**

_____

_____

_____

_____

_____

Appendix C: Survey

44. **3. a) Based on my experience, running automated acceptance tests (with Robot Framework for example) parallel in the pipeline is a good practice** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

45. **3. b) How often automated acceptance tests are ran parallel in the pipeline in the projects you have been working on?**

_____

46. **3. c) If you wish, explain your answers to questions 3 a & b**

_____

_____

_____

_____

_____

47. **4. a) Based on my experience, starting the project with complete pipeline template is a good practice** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

48. **4. b) How often the project has been started with complete pipeline template in the projects you have been working on?**

_____

49. **4. c) If you wish, explain your answers to questions 4 a & b**

_____

_____

_____

_____

_____

Appendix C: Survey

50. **5. a) Based on my experience, having a support team, which enables efficient tools and access rights for development team is a good practice** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

51. **5. b) How often you have had a support team, which enables efficient tools and access rights for development team in the projects you have been working on?**

_____

52. **5. c) If you wish, explain your answers to questions 5 a & b**

_____

_____

_____

_____

_____

53. **6. a) Based on my experience, allowing changes into master branch of version control only by digital code reviews (for example pull requests) accepted by another developer is a good practice** *
Mark only one oval.

- ◯ No experience
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

54. **6. b) How often changes into master branch of version control has happened only by digital code reviews accepted by another developers in the projects you have been working on?**

_____

55. **6. c) If you wish, explain your answers to questions 6 a & b**

_____

_____

_____

_____

_____

Appendix C: Survey

56. **7. Have you encountered any other best practices, which are not described in questions 1-6?**

Questions related to team structures

Release engineering team refers to team members, which are responsible of doing releases and maintaining pipeline.

Testing team refers to team members, who are responsible for writing automated acceptance tests, for example with Robot Framework.

57. **Based on my experience, I consider separate release engineering team important in CD project.** *

Mark only one oval.

- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

58. **Explain your answer relating to this team structure. Which experiences lead to this answer?**

59. **Based on my experience, I consider separate testing team important in CD project.** *

Mark only one oval.

- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

Appendix C: Survey

60. **Explain your answer relating to this team structure. Which experiences lead to this answer?**

61. **Do you have any other opinions about team structures in CD projects?**

Free comment section

In this chapter you may give any comments relating CD or the questionnaire itself.

62. **Are there any positive aspects of CD that you have not yet highlighted?**

63. **Are there any negative aspects of CD that you have not yet highlighted?**

64. **Any other comments?**